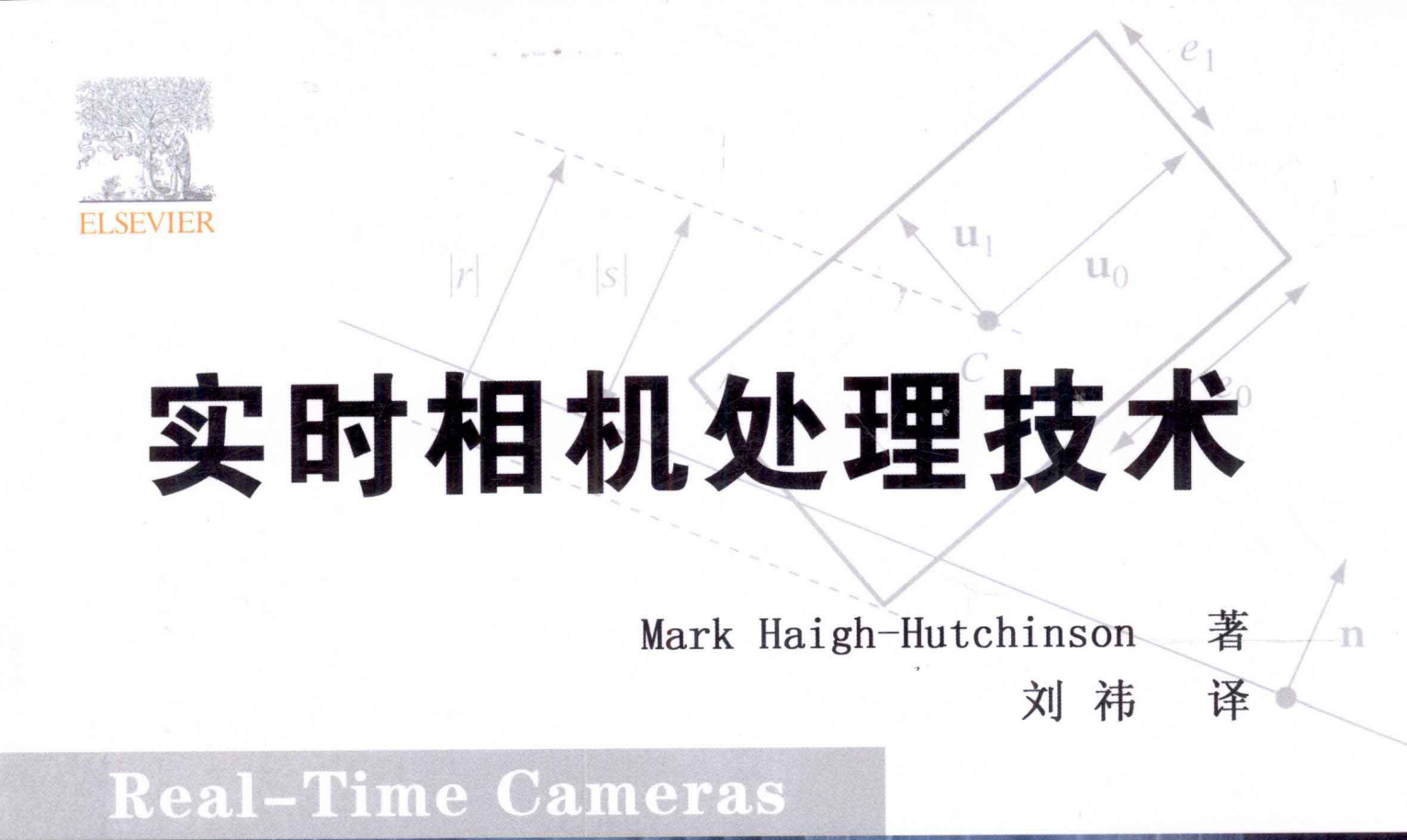


实时相机处理技术

Mark Haigh-Hutchinson 著
刘祎 译

Real-Time Cameras

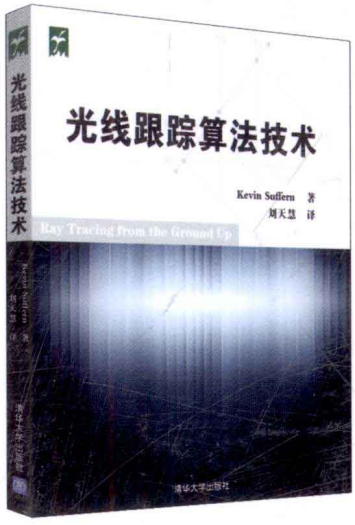


清华大学出版社



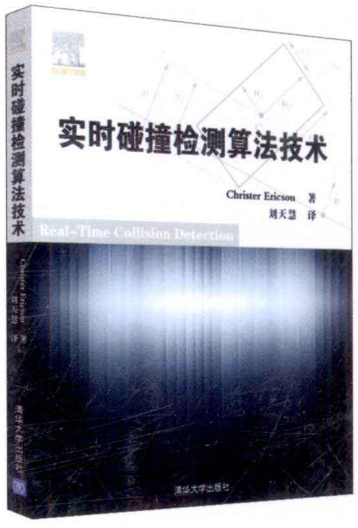
实时相机处理技术

ISBN:978-7-302-27392-9 定价:69.00元



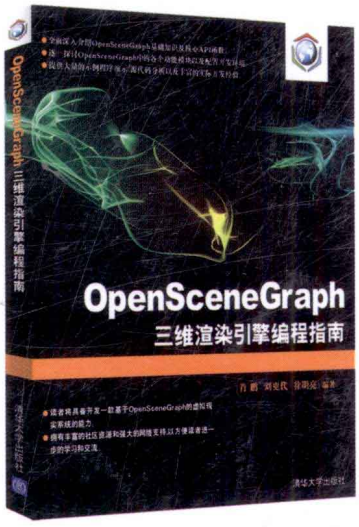
光线跟踪算法技术

ISBN:978-7-302-24821-7 定价:98.00元



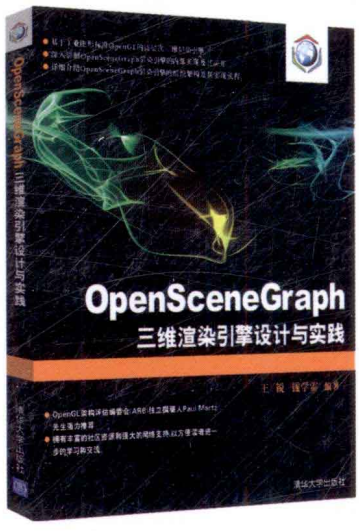
实时碰撞检测算法技术

ISBN:978-7-302-22411-2 定价:52.00元



OpenSceneGraph三维渲染引擎编程指南

ISBN:978-7-302-21303-1 定价:49.80元



OpenSceneGraph三维渲染引擎设计与实践

ISBN:978-7-302-21223-2 定价:45.00元

Real-Time Cameras

ELSEVIER

ISBN 978-7-302-27392-9
9 787302 273929 >
定价：69.00元

实时相机处理技术

Mark Haigh-Hutchinson 著

刘 祎 译

清华大学出版社

北 京

内 容 简 介

本书详细阐述了与实时相机相关的高效解决方案及相应的数据结构和算法，主要包括游戏概述、相机基础知识、电影技术、相机设计、相机设计方案、相机脚本、位置和方向、漫游和遮挡技术、移动和碰撞、相机数学知识以及相机实现方案等内容。此外，本书还提供了相应的算法、代码以及伪代码，以帮助读者进一步理解相关方案的实现过程。

本书适合作为高等院校计算机及相关专业的教材和教学参考书，也可作为相关开发人员的自学教材和参考手册。

Real-Time Cameras, 1e
Mark Haigh-Hutchinson
ISBN: 9780123116345
Copyright © 2009 by Elsevier. All rights reserved.
Authorized Simplified Chinese translation edition published by Elsevier (Singapore) Pte Ltd Press and Tsinghua University Press.
Copyright © 2011 by Elsevier (Singapore) Pte Ltd and Tsinghua University Press. All rights reserved.
Published in China by Tsinghua University Press under special arrangement with Elsevier (Singapore) Pte Ltd.. This edition is authorized for sale in China only, excluding Hong Kong SAR and Taiwan. Unauthorized export of this edition is a violation of the Copyright Act. Violation of this Law is subject to Civil and Criminal Penalties.
本书简体中文版由 Elsevier (Singapore) Pte Ltd.授予清华大学出版社在中国大陆地区（不包括香港、澳门特别行政区以及台湾地区）出版与发行。未经许可之出口，视为违反著作权法，将受法律之制裁。

本书封底贴有 Elsevier 防伪标签，无标签者不得销售。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目（CIP）数据

实时相机处理技术 Real-Time Cameras/（美）哈齐森（Hutchinson,M.H.）著；刘祎译.
—北京：清华大学出版社，2011.12

ISBN 978-7-302-27392-9

I. ①实… II. ①哈… ②刘… III. ①照相机-数据处理 IV. ①TB852.1

中国版本图书馆 CIP 数据核字（2011）第 245154 号

责任编辑：赵洛育
封面设计：刘　超
版式设计：文森时代
责任校对：王　云
责任印制：王秀菊
出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
社　　总　　机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：北京鑫丰华彩印有限公司
装 订 者：三河市兴旺装订有限公司
经　　销：全国新华书店
开　　本：203×260　**印　张**：19.5　**字　数**：520 千字
版　　次：2011 年 12 月第 1 版　　**印　　次**：2011 年 12 月第1 次印刷
印　　数：1～4000
定　　价：69.00 元

产品编号：042042-01

译 者 序

现代交互式实时应用程序（如视频游戏）向用户展示了虚拟世界中动态变化、色彩斑斓的视觉体验。然而，游戏场景、视觉效果、画面人物以及相关操控方案往往与相机系统紧密相关。

长期以来，视频游戏中的相机系统一直未得到妥善处理。高质量的游戏通常会展示华丽的图形效果以及丰富的游戏体验，这常使得某些相机控制系统力不从心。作为观察者与虚拟世界的主要交互途径，相机系统的质量对于玩家的满意度以及游戏体验起到了至关重要的作用。同时，相机系统可视为交互设计过程中的重要组成部分，并可极大地提升观察者的体验感。

针对这一问题，本书详细阐述了与实时相机相关的高效解决方案并涵盖了丰富的内容，主要包括游戏概述、相机基础知识、电影技术、相机设计、相机设计方案、相机脚本、位置和方向、漫游和遮挡技术、移动和碰撞、相机数学知识以及相机实现方案等内容。值得一提的是，本书并非是一本纯理论书籍，除了对相关内容进行了全面、系统的讲解以外，其设计思想、数据结构和算法均辅以对应的代码示例，以帮助读者进一步理解计算方案的实现过程。

在本书的翻译过程中，除刘祎外，翁桂荣、康照丽、孙颖、万立梅、陈丽丽、龙清伟、邰利萍、高影、安娇娇、李霞、刘笑彤、周宏丽、魏晶、李芳芳、张凤欣、高阳、胡艳娇、商婵婵、武变琴、顾庆娟、蒋丽丽、张如英、陈西红和田淑美等人也参与了部分翻译工作，在此一并表示感谢。

限于译者的水平，译文中难免有错误和不妥之处，恳请广大读者批评指正。

译 者

前　言

2006年末，Mark Haigh-Hutchinson（以下简称为Mark H-H）即开始着手编写一本关于实时相机系统方面的书籍，当我得知这一消息后，第一反应便是震惊。作为系列游戏《银河战士》中相机系统的缔造者，以及业界从业者的长期经验，本人坚信，此书作者非Mark H-H莫属。

长期以来，视频游戏中的相机系统一直未得到妥善处理。高质量的游戏通常会展示华丽的图形效果以及丰富的游戏体验，因而这常使得相机控制系统力不从心。例如，游戏画面的不连续现象、延迟、视口抖动、相机对象穿越墙体或方向错误。

因此，拾残局者一定是Mark H-H，他对工作的投入精神、近乎于残酷的完美主义精神以及长期满血的工作状态常具有化腐朽为神奇的力量——这便是我极力推崇Mark H-H撰写此书的原因。

经过短暂交流，Mark H-H便着手于本书手稿的创作工作。在此期间，Mark H-H和我正协同Retro Studios公司的其他人员致力于游戏《银河战士3：堕落》的研发工作，该游戏定义为Nintendo Wii游戏机平台上的旗舰作品。Mark为本书投入了大量的心血，期间，为了得到更多的意见与反馈信息，每章内容完成后，他坚持向我寄送本书的样章，其坚韧不拔的精神令我暗自吃惊。

然而，随后发生的事情令人难以置信。当游戏任务接近尾声时，Mark H-H的消化道系统出现了异常情况；2007年8月，即《银河战士3》发布后不久，Mark H-H的病情不断恶化并被诊断为胰腺癌。

一切都像是一场梦魇，醒来后却身处Mark H-H的葬礼上，时间定格在2008年1月末。

受Mark H-H的妻子Melanie之托，我负责在Mark H-H的葬礼上致悼词，我十分荣幸能有这样一个机会与Mark H-H交谈：

作为Mark H-H的同事、朋友，结识你是我一生的骄傲。

每当思念起Mark H-H的时候，他的奉献精神令我们久久不能忘怀，因而这份思念也就变得愈发强烈。

Mark H-H将一生中最美好的时光、无穷的能量以及洋溢的热情毫无保留地奉献于我们。我们受益于你渊博的知识以及丰富的想象力；你赋予我们源自心底的无私关爱，这份伟大的爱甚至超出了你非凡的智慧。Mark H-H的身后是3部令人难以忘怀的游戏作品、一部足以改变业界现状的学术专著以及朋友间的无尽思念，你的付出令我们无以回报。

Mark H-H温文尔雅的举止和无私奉献的精神将永远活在我们的心中。他的付出源于对Retro以及游戏业的远见卓识，并引领我们抵达理想的彼岸。

请Mark H-H一家接受我对Mark H-H故去的吊慰，请朋友们接受我对Mark H-H故去的哀思。从今天开始，游戏业失去了一位并肩作战的斗士、一位学习的榜样以及一位值得深交的朋友，这对于当今业界而言无疑是一个巨大的损失。

虽然本书最终未能完成，但本书对Mark H-H，甚至是整个游戏界而言，其重要性不言而喻，因此，我们能做到的，只此八字——继承遗志，奋酬君愿。2008年春夏之交，Mark H-H的朋友以及同事相聚在一起，并完成了本书最终的出版任务。期间，每个人无一不沉浸在对Mark H-H的深切缅怀中。

本书的编写工作源于以下人员的通力合作，他们是Ryan Cornelius（软件工程师），Mark DeLoura（视频游戏技术顾问）、Jim Gage（高级软件工程师）、Jack Mathews（技术指导）、Steve McCrea（高级

工程师）、Akintunde Omitowoju（软件工程师）、Eric Rehmeyer（软件工程师）、Marco Thrush（工程师）以及 Paul Tozour（独立咨询师）。

Mark H-H，你将永远活在我们心中。我们将永远铭记你的杰出贡献，回忆一起走过的日子——我们的同事兼良师益友 Haigh-Hutchinson。

在此，我们怀着无比感激的心情呈上本书并借此向 Mark Haigh-Hutchinson 表达最为诚挚的思念。

致谢

下列人员和机构对本书的出版提供了巨大的帮助，他们是 Akintunde Omitowoju、Ryan Harris（合作者）、John Giors、Kynan Pearson、Aaron Walker、Retro Studios、Previous Colleagues、Nintendo 以及 Michael Kelbaugh。

另外，本书编写伊始得到了 Andrew Glassner（由 Morgan Kaufman 出版社引荐）和 Eric Haines 的多方支持，在此向他们表示感谢。

最后，以下人员对本书提供了宝贵的建议和意见，在此一并表示感谢，他们是 Fred Ford、Jim Van Verth、Ed Rotberg、Peter Lipson、Jon Knoles 以及 Ryan Harris。

关于作者

Mark Haigh-Hutchinson 就职于 Retro Studios Inc，该公司坐落于德克萨斯州首府奥斯汀。Mark H-H 于 1984 年便从事视频游戏的设计和开发工作，而他对游戏的痴迷则可追溯到 1979 年。Mark H-H 曾参与了 40 多款视频游戏的开发工作，其饰演的角色也不一而同，包括程序员、设计人员以及项目负责人，其中，较为著名的作品包括星球大战系列游戏，即《星球大战：侠盗中队》、《星战前传 1：极速飞车》、《星球大战：帝国的阴影》以及《星球大战：绝地大反攻》。除此之外，其他作品还包括《僵尸邻居》、《萨姆和马克斯：开始流浪》以及《太空高速公路》。Mark H-H 主要负责以下作品中相机系统以及玩家控制方面的工作：《银河战士 3：堕落》、《银河战士 2：回声》以及获奖作品《银河战士》。

Paul Tozour

概　述

视频游戏以及其他交互式实时应用程序在展现虚拟场景的视见画面时应符合环境特征并满足相应的美学条件，与此对应的相关内部机制则称为虚拟相机系统。作为观察者与虚拟世界的主要交互途径，相机系统的质量对于玩家的满意度以及游戏体验起到了至关重要的作用。拙劣的相机系统将在很大程度上削弱应用程序的图形效果，抑或交互机制无法满足相关的游戏需求。另外，视频游戏分析过程中常会发现诸多问题，如拙劣的取景操作、相机在场景中的漫游问题等，这一类问题往往会对游戏的整体评价产生很大的负面影响。另一方面，设计良好的相机系统使得复杂的交互设计过程变得一目了然，并以一种和谐、易于理解的方式展现当前虚拟场景世界。然而，针对交互式相机系统的设计人员以及开发人员而言，相关资料并不多见。

对于这一类重要且常被忽视的领域，本书旨在向设计师以及开发人员提供与此相关的扩展内容。尽管传统的电影技术与虚拟相机系统存在交集（如非交互式电影视频序列），但视频游戏的动态变化特征以及其他交互式实时程序均要求采用截然不同的处理方案。与传统的电影工业相比，交互式程序设计的出现则可视为一类新生事物，其中，实时相机的设计规范依然处于成型期中。然而，现有信息足以形成一套基本的设计规则，且与游戏类型以及表现风格无关。

针对于此，本书的设计理念将秉承以下原则：强调虚拟相机系统的设计过程，而弱化相机系统的存在（针对观察者而言）。

如果相机系统可正确地展现虚拟场景环境，并使得玩家沉浸于游戏体验过程中，则该系统即可视为一类相对成功的相机系统。

相机系统设计在程序设计过程中占有重要的地位，因而理应在其初始设计阶段、原型设计阶段以及最终的产品实现阶段占用较大的开发资源；针对于相机的开发过程，其重要性将贯穿于全书中的大部分内容，以使相机系统的质量更上一层楼。

除了表达设计师或美工师期望中的非交互式（或电影视频）序列，通过对相机位置、方向以及其他属性加以适当管理，相机系统可在交互式序列中极大地增加观察者的体验感。通用相机系统的设计以及实现过程较具挑战性，且应以一致、和谐以及适宜的方式展现交互环境的多样性变化以及相关要求。

需要注意的是，针对玩家与虚拟场景之间的观感，相机位置和方向的简单变化均会产生较为明显的影响。另外，场景内角色或对象的相对尺寸以及位置数据同样会使观察者眼中的相关对象产生显著的变化。这里，若观察者无法有效地辨识虚拟世界对象（主角或玩家角色）与场景元素之间的交互关系，则相机系统可视为失败之作。

因此，与实时相机系统相关的第二条原则是：交互式应用程序的表现方式直接影响到相机系统的设计和实现水准。

针对于此，本书将着重强调实时交互式相机系统的实现方案。这里，在实际交互过程中将对非交互式电影视频序列以及相机的各类用途加以严格区分。电影技术涵盖了诸多内容，尽管如此，设计人员依然需要掌握相关的背景知识。需要注意的是，针对于实时环境，上述技术应谨慎使用。

此处，虚拟相机系统由以下两个主要部分构成，即交互设计以及工程设计。针对于前者，对应系统多由专业设计人员进行开发，且需要对虚拟环境、引擎功能以及相关工具集进行反复调整；而后者

往往取决于工程人员的技术能力，并对现有技术加以适当的调整，抑或在满足开发周期、性能等相关需求的前提下创建新型相机系统。无论如何，工程任务应深入考察设计人员所制订的交互需求条件。另外，相机系统可视为交互设计过程中的重要组成部分，并应极大地提升观察者的体验感。

本书大部分内容将集中于视频游戏以及该环境下相机系统的应用方案。需要注意的是，相机设计以及用户操控在各类交互式计算中均饰演着重要的角色，而非仅仅是游戏应用程序。实时医学成像、虚拟现实系统、机器人学以及其他相关领域均需要相应的虚拟环境漫游操作，本书所涉及的大多数内容同样适用于这一类领域。实际上，传统的电影拍摄手法也大量地借鉴了游戏中的相关技术，如视频预览技术、镜头构成以及特效规划。当然，不同领域之间的关键因素主要体现在交互性特征方面，因此，这将引入第三条规则：实时相机系统将对虚拟场景交互过程中玩家的操控感产生潜移默化的影响。

针对虚拟场景投影视图及其在场景中的表现方式，玩家对此通常难于理解，因此相机系统与玩家操控之间的交互过程将变得十分重要。另外，观察者与游戏场景投影视图之间的分离状态通常难以避免。然而，这一类元素需要在开发过程中予以全面考察，进而避免潜在问题所产生的负面影响。

读者对象

本书涉猎广泛，相关内容包括基于游戏相机的设计和实现方案以及其他实时应用程序。需要注意的是，一款成功的相机系统常需要设计师、美工师以及程序员之间的通力合作方可实现。在游戏体验过程中，应对上述各条规则加以逐一考察，进而满足游戏设计中的约束条件、美学需求以及其他技术限制条件。

本书针对相机设计理论进行了较大篇幅的介绍，且与游戏美工师的日常工作紧密相关；当然，这对实现相机系统或玩家控制系统的程序员而言也大有裨益。需要说明的是，本书并非是一本电影理论书籍，但设计人员以及美工师理应对此加以关注，并将其与实时游戏技术实现有机的结合。另外，本书还将针对场景构建方案进行深入讨论，以避免相机系统实现过程中的潜在问题。无论是游戏抑或其他实时交互式应用程序，程序员通常负责相机系统的实现过程——这要求程序员熟悉相关规则，进而为设计步骤打下坚实的基础。

阅读方式

本书包含 3 部分内容，即核心概念、设计原理以及相机工程。其中，核心概念部分提供了与实时相机（如视频游戏）应用方式相关的基础内容。设计原理部分则对此实施进一步扩展，相关内容包括需求条件以及相机设计过程中的既定方案。相机工程部分则对虚拟相机中的各类属性加以考察，包括相机系统实现过程中的理论和实现方案。

另外，本书各章均涉及以下两部分内容，即设计过程以及相关技术方案。其中，“设计过程”部分主要讨论相机设计过程中的理论知识，并在必要时辅以少量的技术观点。“技术方案”部分将作为重要内容加以介绍，必要时，读者还应参考相关资料以获取进一步的信息。理解相机系统背后的理论和需求条件对程序设计人员而言通常大有裨益，但通常情况下，设计人员和美工师往往对此部分内容更感

兴趣；相应地，“技术方案”部分则对相机系统的实现过程进行深入分析，其中包括算法、伪代码以及某些 C++示例代码。当然，这也要求读者具备某些与程序设计相关的背景知识。同时，与游戏开发相关的某些实际问题将贯穿于本书的全部章节。

因此，本书的前几章将主要介绍原理性内容，后续内容则围绕具体的技术实现方案加以讨论，进而完善读者的知识体系。

第 1 部分：核心概念

第 1 章（游戏概述）阐述了典型视频游戏的循环更新操作，以及相机在游戏环境中的各项功能。第 2 章（相机基础知识）讲述了实时相机的基本定义及其与真实相机设备之间的差异；除了游戏相机设计规则概览之外，本章还简要介绍了相机属性的应用方式，并用于生成游戏场景的期望视见画面。虽然本书的重点在于实时相机，但事实证明，电影理论以及相关规则依然十分重要。当实时系统需要处理电影视频序列时更是如此，无论是常规应用或镜头跳转（亦即，类似于电影的非交互式视频序列）。第 3 章（电影技术）阐述了电影技术规则及其在实时应用程序中的应用方式。

第 2 部分：设计原理

相机系统的设计过程应着重对期望特征集以及表现方式加以考察，进而满足观察者的交互需求。第 4 章（相机设计）讲述了定义相机系统特征集时的诸多相关因素。针对相机系统设计，产品设计过程通常不可或缺。对于相机设计过程，存在某些既定方案可供使用，相关问题可对相机系统开发工作提供较好的指导作用。在交互设计的初始阶段，相机设计往往作为重点内容加以考察，且暂不考虑游戏类型。需要注意的是，特定的游戏类型往往决定了其表现方式。第 5 章（相机设计方案）针对特定的游戏风格，探讨与此相关且较为常见的处理方案。需要指出的是，相关方案不仅局限于视频游戏中且对其他应用程序同样有效。另外，针对单显示设备，本章还讨论了单一玩家和多玩家表现方式之间的差异。

在交互式或非交互式序列中，相机脚本机制则用于控制实时相机行为变化的开启和定时操作，这一类变化常对应于预置事件且出现于应用程序执行过程中的既定时刻。另外，相机脚本还用于定义、反馈动态变化的交互元素，进而可全程提供适宜的虚拟场景视见画面。针对相机和玩家控制脚本的处理方案以及相关功能项，第 6 章（相机脚本）则对此进行了深入探讨。

第 3 部分：相机工程

鉴于相机的期望位置和方向直接决定了虚拟世界的视见画面内容，因而可视为相机系统的主要组成部分。第 7 章（位置和方向）根据游戏体验以及美学需求，将主要讨论上述各属性的定义方式。虚拟相机通常需要在复杂的场景环境中实施移动操作，因此，确定相机在场景中的遍历方式并以适当的

方式平衡（展现目标对象时的）游戏体验需求将是一项十分艰巨的任务。第 8 章（漫游和遮挡技术）针对于漫游和遮挡技术，分别阐述了动态和预定义方案，亦称作漫游操作，进而可确保虚拟场景元素（如主要角色或玩家角色）位于可见位置，且无须考虑相机的移动方式。另外，本章还探讨了遮挡问题的多种定义方式以及避免此类问题的相关方案。第 9 章（移动和碰撞）介绍了相机在虚拟场景中的移动方式，且不失运动过程中的连续性和平滑性。然而，此类相机移动可能会导致与场景环境元素之间的碰撞现象。因此，本章还涵盖了常见的碰撞躲避机制、碰撞检测及其对应处理方案。实时相机的实现过程需要运用到大量的数学知识，因此第 10 章（相机数学知识）包含了数学应用方案，进而对相机系统中的相关问题进行求解。第 11 章（相机实现方案）叙述了相机系统实现方案中的体系结构问题，并可根据交互需求条件动态地调整相机行为。

附录

电影技术包含大量的术语并已形成一门独特的语言，且应用广泛、易于理解。然而，实时应用程序中的相机应用并未达到此类高度。附录 A（术语表）包含了与相机定义相关的基础内容，并以此彰显不同规则间的相互关系。附录 B（辅助材料）包含了读者可能感兴趣的相关资料，其中包括视觉感知及其与实时相机设计之间的关系。

参考文献

与实时相机相关的参考文献目前尚不多见，但相关数字仍处于不断增长中。实际上，源于 20 世纪 80 年代的机器人学业已掀起了人工智能的热潮，尤其是复杂场景环境中的漫游技术。类似地，关于用户界面以及相机控制，医学成像领域对此均有所应用。近些年来，SIGGRAPH（网址为 www.siggraph.org）发表了多篇与相机应用相关的论文，且内容多与非交互式电影技术以及拍摄技术相关；近期，“游戏开发者大会（网址为 www.gdconf.com）”也对实时相机系统加深了关注力度，其中包含了相关的参考资料（以及在线资源）并根据具体应用加以分类。

其他辅助资源

读者可直接访问本书的合作网站 www.realtimecameras.com 以获取更多资源，其中包含了一款基于 Windows XP 和 OS X 操作系统的相机系统源代码。

针对于相机理论以及相机系统开发过程中的基本原理，相机系统应能够提供较好的测试环境。另外，该网站还提供了电影演示作品、源代码、现有游戏作品中相机设计的分析资料以及与相机系统设计、实现相关的其他资源。

最后，读者可对本书发表自己的意见或建议；除此之外，如果读者对交互式相机系统拥有独到的见解，亦望不吝赐教，具体联系方式参见本书合作网站。

目　录

第 1 部分：核心概念

第 1 章　游戏概述 3
实时应用程序 3
游戏系统 3
游戏的循环更新结构 4
输入 6
对象的思考步骤 6
对象的移动 7
消息机制 7
相机 8
相机后处理 8
渲染器 8
常见的性能问题 9
相机系统 9
控制操作 10
脚本设计 11
本章小结 12
第 2 章　相机基础知识 13
真实的相机系统 13
用途和属性 14
电影投影 15
游戏相机 15
基本技术 16
相机系统 17
游戏相机 17
表现风格 17
相机行为 17
第一人称相机和第三人称相机 17
电影相机和交互式相机 18
观察位置 18
期望位置 18
方向 18
偏航角 19
仰角 19
旋转角 19
期望方向 19
旋转操作 19
视见体 19
视口 19
视域 20
宽高比 20
刷新率和帧速率 21
安全区和安全帧 21
刷新率和运动模糊 22
显示撕裂 23
玩家角色人物 23
目标对象 23
插值计算 23
投影 24
视差 24
相机转换 24
相机约束条件 24
移动约束条件 25
方向约束条件 25
相机的表现风格 26
正交表现方式（2D） 26
透视表现方式（3D） 27
混合表现方式（2.5D） 28
相机行为 29

视频相机 29
第一人称/视点 30
第三人称/分离视角 31
后定位 33
肩部定位 33
脚部可见 33
远距离定位 34
角色相关 34
静态相机 34
路径定位 34
混合 FP/TP 相机 35
预视相机和反馈相机 36
预测人物的移动方式 37
预测相机的路径 37
路径搜索方案 38
数据驱动与规划方案 38
方向预测 38
视见生成 38
视锥体 40
视见转换 40
投影转换 41
玩家控制操作 41
本章小结 42
第 3 章 电影技术 43
术语 43
标准电影术语 44
移动拍摄 44
摇动镜头 44
镜头倾斜 45
跟踪拍摄 45
景深 45
淡入/淡出效果 46
镜头眩光效果 46
旁跳镜头 46
插入镜头 46
跳接镜头 47
180° 规则/动作作用线 47
镜头视点 47
升降镜头 47
特写镜头 48
反转镜头 48
30° 规则 48
强迫透视 48
实时电影技术 48
电影 48
交互式电影技术 49
电影摄像机 50
摄像机的移动 50
摄像机的方向 51
动态生成画面/回放摄像机 51
再现型摄像机/关键帧摄像机 52
脚本型摄像机 52
动态生成画面 53
结束画面 53
开始画面 54
回放相机的效果 54
重定向 54
摄像机抖动效果 54
相机的摇动效果 55
视域/景深 55
脚本机制 55
剪辑操作 56
镜头选取 57
取景/合成操作 57
场景转换 57
跳转镜头 58
淡入/淡出转换 58
圈入/圈出转换 58
划接 58
视口转换 59
工具 59
本章小结 60

第 2 部分：设计原理

第 4 章　相机设计 63
交互式 2D 相机系统 63
期望特征 64
确定视见内容 65
2D 电影相机系统 66
交互式 3D 相机系统 67
透视投影 68
平行投影 68
3D 电影视频相机系统 70
2.5D 相机系统 71
游戏设备 71
宽高比 72
调整尺寸 72
镜头的摇动和扫描 72
拉伸效果 72
相机设计 72
游戏以及技术需求 73
设计概览 73
考察高层设计目标 73
评估玩家角色行为 74
确定场景环境范围 74
定义基本的相机行为 74
特定区域范围内的场景需求 74
相机系统的技术评估 75
游戏设计原则 78
本章小结 80

第 5 章　相机设计方案 81
基于游戏类型的相机设计方案 81
第一人称射击/冒险类游戏 81
相机的定位方案 81
角色类/动作类冒险游戏 84
3D 平台 86
角色饰演类游戏/在线多人游戏 88
滚屏机制 88
竞技类游戏 89
单玩家竞技类游戏 90
双人或 4 人竞技类游戏 90
团队竞技类游戏 90
室内场地类游戏 91
室外竞技类游戏 91
抽象功能 92
赛车类游戏 92
地面（或表面）机动设备 93
即时战略游戏 93
飞行模拟 94
具有真实效果的飞行模型 94
非真实相机模型 95
人工水平仪 95
运动感 95
冒险类游戏 95
拼图类/Party 类/棋盘类游戏 96
格斗类游戏/近战游戏 97
多玩家相机处理方案 98
单屏技术 99
分屏技术 100
全屏与分屏之间的转换 101
分屏与全屏之间的转换 101
本章小结 102

第 6 章　相机脚本 103
相机脚本概述 103
脚本类型 103
脚本语言 104
基于文本的脚本语言 104
通用编程语言 104
自定义脚本语言 105
有限状态机 106

可视化脚本语言 107
脚本语言中的问题 108
事件消息机制 108
混合脚本方案 110
脚本对象 110
对象关系 111
相机脚本机制 115
相机脚本方法 115
相机控制 116
第三人称脚本机制 116
第一人称脚本机制 116
非交互式电影脚本 118
经验方法 118
本章小结 121

第 3 部分：相机工程

第 7 章　位置和方向 125
坐标方案 125
世界空间 125
相机（视见）空间 125
屏幕空间 126
局部（角色）空间 126
对象相关 126
期望位置 126
第一人称相机定位方式 127
第三人称相机定位方式 127
期望位置的确定方法 128
静态相机 129
伺服/跟踪操作 130
基于世界坐标系统的偏移量 130
基于世界坐标系统的角偏移量 131
基于世界对象 131
局部偏移量 132
局部角偏移量 132
相对于角色的偏移量 132
相对于角色的角偏移量 133
相对于对象的偏移量 133
路径 134
路径类型 135
线性类型 135
圆形/椭圆形/螺旋形路径 135
样条 136
方向 144
方向表现方法 145
欧拉角 146
转换矩阵 146
轴角 146
四元数 147
期望方向的确定方案 147
固定方向 147
跟踪目标对象或位置 148
视见偏移相对于角色的偏移量 148
锁定视点位置 149
目标对象位置预测 149
对象取景 150
视见位置的确定方案 150
美学条件与游戏体验需求 150
回放相机 151
视域 151
角色的空闲状态 151
自动方向操控 152
第一人称相机与第三人称相机之间的转换 153
第三人称相机与第一人称相机之间的转换 153
重定向方案 153
旋转操作的应用 154
均匀角速度 154
加速和减速 154

角速度阻尼效果 154
全视角阻尼效果 154
降低突然转向的发生几率 155
第三人称自由视角 158
玩家移动 158
自由视角定向方案 158
常见方向问题 159
万向节死锁 159
垂直转向 159
倾斜操作 160
方向噪声 160
快速方向变化 160
玩家角色的视锥体剪裁 161
本章小结 161
第 8 章　漫游和遮挡技术 163
将相机视为 AI 游戏对象 163
漫游技术 164
动态漫游技术 164
预定义漫游技术 168
流场 171
影响力地图 171
势场 171
环境遮挡 172
如何看待环境遮挡问题 172
场景遮挡确定方案 173
光线投射技术 173
空间体投射 174
渲染技术 174
遮挡预测方案 175
预测视点位置 175
视线 175
处理视线问题 176
预测方案 176
图形变化 176
获取期望位置 177
瞬移/跳转操作 177
保持控制方向 177
生成路径 177
跟随角色移动 178
凸起地形 178
回避纵向柱状体 179
预计算方案 179
防止丢失 LOS 179
瞬间移动 180
淡化遮挡几何体或对象 180
无须渲染遮挡几何体或对象 180
渲染图形元素且忽略遮挡现象 180
可见性预计算方案 180
安全检测 181
安全检测方案 182
本章小结 182
第 9 章　移动和碰撞 183
相机移动序列 183
人物移动 184
第一人称相机 184
第三人称相机 184
移动方案 186
瞬间移动 186
锁定相机 186
比例控制器 187
物理模拟 188
弹性机制（弹簧） 189
PID 控制器 189
圆周运动 190
插值计算 191
平滑效果和阻尼技术 192
阻尼技术 192
移动过滤器 193
移动约束条件 194
纵向移动约束条件 194
渲染几何体的接近程度 195
距离约束条件 195
平面约束条件 196
表面约束条件 196
空间约束条件 196
玩家操控方式 197

相机位置的控制方案 197
观察相机 198
相机移动确认方案 198
位置控制约束条件 199
相机位置控制方案 200
相机方向控制 201
第一人称相机 201
第三人称相机 201
相机的自动定位和定向方案 202
基于玩家控制器的反馈 202
基于游戏体验需求的反馈 203
调试相机的操控方式 203
相机碰撞 204
相机碰撞的重要性 204
碰撞检测的必要性 205
碰撞确定方案 206
对象碰撞 206
环境碰撞 206
碰撞图元 207
碰撞几何体设计 207
避免紧凑型封闭空间 208
向相机提供足够的空间 208
陡峭的阶梯 208
突兀地形 208
低矮的天花板 208
入口 208
采用简单的碰撞网格 209
针对相机的独立碰撞网格 209
圆角对象 209
碰撞决定方案 209
保持静止 209
部分移动 210
移动变换 210
镜头跳转 210
禁用碰撞检测 210
避免相机碰撞 211
玩家的未来位置 212
预测潜在碰撞 212
碰撞预测 213
对象排斥 213
朝向相机移动 214
第 10 章 相机数学知识 215
基础相机数学 215
相机位置 215
相机方向 215
相机移动 216
渲染操作 216
通用相机数学 216
相机问题以及修正方案 216
常见数值技术 216
相机的视见位置 217
避免倾斜旋转 217
降低突然转向的发生几率 218
世界空间与屏幕空间之间的转换 218
四元数 221
指数曲线 222
比例曲线 222
球体线性插值计算 222
超越函数 223
分段样条曲线 223
弹性机制（弹簧） 223
数字滤波器 224
低通滤波器 224
高通滤波器 225
带通滤波器 225
有限脉冲反应滤波器 225
无限脉冲反应滤波器 226
样条曲线 227
相机样条应用 227
三次多项式 228
样条类型 228
线性类型 229
分段式 Hermite 曲线 229
Catmull-Rom 样条曲线 230

圆角 Catmull-Rom 样条曲线 231
Kochanek-Bartels 样条曲线 232
Bezier 曲线 232
均匀三次 B 样条 234
非均匀有理 B 样条 234
连续性 235
样条定义 235
样条计算 235
控制点生成方式 236
参数化弧长 236
全部样条长度 236
最近位置 236
样条编辑 237
插值计算 237
插值的含义 238
相机属性插值计算 238
位置插值计算 239
方向插值计算 239
方向插值特征 241
侧旋插值计算 242
FOV 插值计算 242
视口插值计算 243
玩家控制插值计算 243
第一人称相机 244
第三人称相机 244
插值选择方案 245
线性插值 245
源值 245
目标值 246
插值算法 246
分段式插值 246
插值方案 247
线性时间插值 247
参数函数 248
球体线性插值计算 248
潜在插值问题 248
美学问题 248
数值问题 250
插值中断 251
切换过程 251
切换过程中的位置数据 251
切换过程中的方向数据 252
相机数学问题 252
浮点精确度 252
使用 Epsilon 值 252
编译器偏差 253
硬件 FPU 偏差 253
向量标准化操作 253
矩阵累积浮点误差 253
定期修复数值 253
第 11 章　相机实现方案 255
游戏引擎体系结构 255
游戏更新循环结构 255
游戏系统管理器 256
时间间隔 257
输入处理机制 257
相机系统体系结构 257
视口管理器 258
视口 259
视口切换 259
渲染管理器 260
相机管理器 260
相机更新循环结构 261
提示管理器 261
震动管理器 262
相机震动操作 262
隆声/力反馈效果控制器 262
游戏相机 263
继承的相机行为 263
基于组件的相机行为 264
电影视频相机 264
调试型相机 265
脚本机制实现方案 265
相机脚本对象 265
相机提示对象 265

触发体.....266
渲染/光照效果.....266
消息延迟.....266
序列事件计时器.....266
通用路径.....267
脚本逻辑排序.....267
消息机制.....267
优先权.....267
插值计算.....268
性能问题.....268
分摊机制.....268
预处理机制.....269
开发工具.....269
场景编辑器.....269
相机碰撞网格.....270
相机调试技术.....270
交互式调试技术.....271
独立型调试相机.....271
控制游戏刷新率.....272
通用相机状态.....272
相机属性的视觉表现.....272
属性滞后现象.....273
移动约束条件.....273
视线.....273
基于行为的渲染操作.....274
脚本调试.....274
数据日志.....274
游戏回放机制.....275
本章小结.....275
参考文献.....277
AI 和漫游技术.....277
电影技术.....277
相机设计.....278
游戏设计.....278
图形学和渲染技术.....279
数学.....279
其他资源.....280
在线资源.....280
附录 A 术语表.....281
附录 B 辅助材料.....291
视觉感知.....291
移动敏感度.....291
刷新敏感度.....291
色彩敏感度.....292
深度感.....292

第 1 部分：核心概念

本书第 1～第 3 章讲述了实时相机系统的描述和设计方案，以及基于设计元素和实现细节的常见术语。其中，第 1 章对游戏应用程序进行了综合概述并讨论了相机系统的行为方式；第 2 章确定了基本架构，进而定义游戏相机及其游戏场景视见内容的生成方式；第 3 章介绍了与电影技术相关的基本概念及其与实时应用程序之间的应用方案。

第1章 游戏概述

本章主要阐述实时应用程序中与相机系统相关的各项内容，并包含了较为常见的视频游戏循环更新过程。另外，本章还将对更新游戏状态的典型机制以及相机系统与其他游戏系统之间的整合方案进行深入讨论。如果读者对此颇有心得，可直接跳过本章的阅读。

实时应用程序

现代交互式实时应用程序（如视频游戏）向用户展示了虚拟世界中动态变化、色彩斑斓的视觉体验。当然，大多数用户对于这一类应用程序的开发复杂度以及难度也有所耳闻。在令人炫目的视觉场景背后，实时应用程序体现了问题的多样化，例如，程序需符合高效性并满足用户的体验需求。游戏则将这一点体现得淋漓尽致：场景世界的快速更新（此处，实时二字实至名归）、玩家的反馈处理以及显示设备的连续、无闪烁效果。除此之外，游戏场景世界还应忠实地还原设计者的初衷且不会令玩家产生无所适从的感觉。但最为重要的一点是，游戏应体现出应有的娱乐性。

游戏系统

在实时应用程序的设计以及实现过程中，针对上述较为棘手的问题，最为有效的解决方案是将问题划分为多个组成模块，每个组成模块应包含适宜的可控粒度。在视频游戏中，上述组成模块常称为游戏系统，并操控当前应用程序的不同功能，亦即游戏引擎。较为典型的游戏引擎包含下列内容：

- ☑ 资源管理（内存使用、游戏状态的存储与恢复、模型管理、动画管理、纹理管理、关卡数据加载等）。
- ☑ 玩家控制（如将控制器硬件数据转换为游戏命令）。
- ☑ 物理模拟（作用于游戏对象之上的物理力学及其效果的数学表达方式，如对象的碰撞以及分解）。
- ☑ 游戏对象逻辑（该处理过程定义了游戏的博弈机制）。
- ☑ 图形用户界面（玩家获取游戏信息的视觉界面）。
- ☑ 相机和视口管理（控制游戏场景世界中视觉画面的生成）。
- ☑ 网络接口（例如，与外部实体间的通信，其中包括游戏或硬件设备、匹配服务等）。
- ☑ 人工智能（该机制定义了游戏对象逻辑、自适应性以及针对游戏事件或玩家输入状态的预置或动态反馈）。
- ☑ 游戏事件管理（针对游戏元素中设计者所规定的操作序列进行管理）。

- ☑ 渲染系统（生成游戏场景视觉画面的可行方案）。
- ☑ 对象间的通信（常出现于事件管理以及游戏对象逻辑中）。
- ☑ 音频管理（音效处理过程，包括简单的音频回放以及空间定位功能）。
- ☑ 任务管理（分配相关任务至多处理器、线程管理等）。
- ☑ 其他内容。

某些系统则关注特定的功能，例如相机系统、玩家控制系统以及脚本系统，这一类系统的交互方式以及整合方式多种多样并可加深玩家对游戏的体验。渲染系统与相机系统紧密相关，但本书并不打算对其进行过多的介绍，具体内容可参考[Foley90]和[Akenine-Möller02]。[Eberly04]则对游戏体系结构进行了深入分析。

技术要点

针对相机系统的实现方案，读者可参考第 11 章以获取更为详细的内容，此处仅对其进行简要的描述。上述游戏系统均包含各自的实现方案，有时称之为管理器。游戏系统的实现方式之一便是采用继承结构，其中，顶层游戏管理器或系统管理器起到全局调用的功效。同时，该管理器通常包含某一循环更新结构，并作为其他管理器的中心访问点。除此之外，该管理器还将记录更新之间所流逝的时间值，并在必要时将该值提供于其他管理器。各管理器将根据游戏的特定功能加以组织和实现，相机管理器便是其中一例。相机管理器可视为相机系统在游戏中的实现方案，并提供了相关接口以供其他游戏系统获取与当前相机有关的有效信息、渲染器所需的基本信息等。各个管理器执行其各自逻辑，且在每次循环更新过程中针对游戏对象至少进行一次更新操作。图 1.1 显示了典型的游戏系统概览图以及系统间的通信关系。

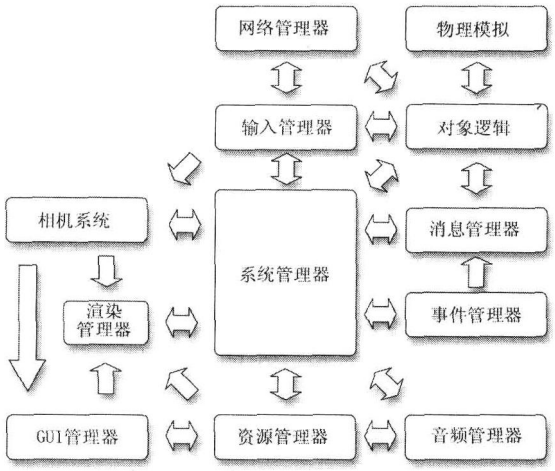


图 1.1　游戏系统管理器的概览图以及系统之间的通信关系

在讨论系统规范之前，首先依次介绍更新和显示实时游戏时所用到的相关解决方案。如果读者对此较为熟悉，可直接阅读“常见的性能问题”一节。

游戏的循环更新结构

视频游戏在操作过程中将会重复执行一系列的操作命令，相关动作将会划分为多个步骤。各个步

骤中的行为将根据不同的游戏需求产生较大的变化（将各个步骤细分为更小的部分），但相关步骤的顺序通常保持不变（尽管有时会省略某些步骤）。这里，全部操作步骤序列称为循环更新结构。另外，根据硬件以及软件环境，循环更新结构中的相关步骤也会不尽相同。

此处，网络通信便是一例。针对于多玩家游戏，通过网络连接接收（或发送）玩家的输入或位置信息有时是十分必要的，这将在输入步骤之后（或之中）引入网络操作步骤；而某些游戏机硬件设备则需要在显示设备尚未更新时（即场消隐期）并在某一较小时间段内更新图形数据。另外，也可采用独立的步骤与上述操作同步执行。

在游戏的运行过程中，上述步骤的执行顺序可能会稍有不同。例如，当游戏暂停或显示相关菜单时（游戏正式开始之前的加载界面便是一例），一般不会执行对象的移动及其逻辑操作。

针对于全部可能性，存在多种对应的排列方式，图 1.2 显示了循环更新序列的流程图。

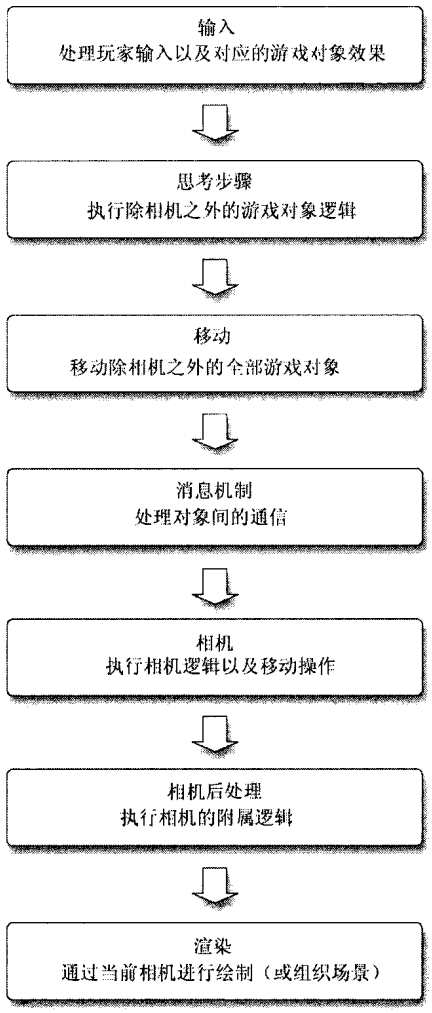


图 1.2　典型的游戏循环更新结构

在游戏的运行过程中，循环更新结构重复执行上述步骤。根据特定的游戏状态，某些步骤可能会稍有变化（如弹出选择菜单时）。下面依次对相关步骤加以简要的描述。

技术要点

需要注意的是，在游戏的循环更新结构中，相机逻辑操作一般在大多数游戏逻辑之后方可执行，该顺序可确保相机的位置以及方向能够正确地反映出游戏对象的最新位置，当相机对象属性继承自其他游戏对象时尤是如此。然而，基于当前操作顺序，仍然存在某些与此相关的依赖问题。例如，如果消息处理阶段导致游戏开启了影片播放序列，则相机逻辑将运行于新状态下，而之前的游戏对象仍然处于原逻辑状态下。

这也意味着，相机的更新操作将超前于其他游戏对象。当游戏对象逻辑（如场景世界中的位置数据）在影片播放序列之前进行初始化时将产生严重的问题。

针对于此，存在多种解决方案。一种解决方案是延迟影片播放状态，直至出现相机逻辑以及渲染操作（若并行执行，则延迟至初始化状态之后）。作为循环更新结构中的最终步骤（此时，渲染器已初始化完毕），该方案将包含全部与相机相关的消息处理内容。另一种解决方案是将消息处理延迟至循环更新结构中的最尾处。

输入

输入操作步骤负责读取控制设备的各项参数值（或与其他资源进行比较。此处，硬件访问源于中断控制器），并通过游戏引擎逻辑将其转换为读取游戏中的命令，如玩家移动、游戏暂停、选择相关菜单等。通常情况下，考虑到硬件的局限性以及游戏逻辑和玩家移动每帧只执行一次，因而控制器在每次循环更新过程中也仅读取一次。然而，某些游戏将在同一循环更新结构中对玩家的输入数据进行多次分析。

与循环更新过程中的单次读取相比，上述方案仅当硬件以较高频率提供数据信息时才切实可行（通常情况下也十分有效）。此时，可准确获取基于玩家期望动作的相关信息，这一类信息可对玩家的移动进行有效的预判，抑或使得玩家的运动以及控制逻辑可在单次更新过程中执行多次，进而获取更为精准的移动和控制操作。此处，即使游戏的刷新率较低（如网络延迟），玩家依然可以获取较好的控制效果。

技术要点

游戏系统管理器（此处为游戏对象管理器）通常用于控制游戏对象逻辑的执行，包括优先权以及对象间的依赖关系。针对游戏逻辑的分摊机制，游戏对象控制器可采用两种处理方案，即协同机制和优先多任务机制。在协同机制的某一既定更新过程中，各个游戏对象受限于处理器的时间值，并在事发时向游戏对象控制器自动返回控制项。因此，在确定游戏策略之前，游戏对象需要多次运行其自身逻辑内容。

除此之外，或许作为一种附加的分摊步骤，游戏对象管理器可在某一既定循环更新结构中负责确定哪一类游戏对象执行其自身的逻辑内容（通常称作负载平衡）。这需要开发人员了解不同游戏对象之间的相对性能开销，需要说明的是，性能开销的计算过程往往是即兴之作，随着游戏执行过程的不断深入，通常会伴随着预计算、动态分析以及调整等内容（例如，针对特定实例类型的游戏对象个体，记录其所需的平均时间量）。

在优先多任务机制中，游戏对象管理器将在预决策期后中断某一特定游戏对象的执行任务，因此需要预先存储该游戏对象的状态数据，并在下一次执行之前进行恢复。该方案的难点在于：在执行的中断点处，游戏对象状态的有效性无法得到保证（例如，跟踪对象的位置数据在其逻辑更新期间可能会产生变化）。

虽然对象的逻辑开销可能会分摊于多个循环更新中，但需在每次循环中对该对象的位移以及碰撞进行测试。另外，并行处理以及多线程任务机制还将会大大增加任务的复杂度并产生顺序依赖等问题。

对象的思考步骤

对象的思考步骤占据了游戏逻辑的大部分内容，并包含由输入步骤所提供的玩家逻辑、游戏对象

的人工智能（AI）逻辑、游戏事件（或其控制操作）等。该步骤属于处理器敏感型事件，针对于后续循环更新，其主要工作集中于获取正确的游戏对象性能分摊机制。AI逻辑即是如此，每次执行循环更新时，游戏对象（如敌方角色）都需要相应地调整其行为方式。

通常情况下，敌方对象的反馈级别无须遵循真实状态。相比较而言，在真实世界中，根据动作行为的不同特征，人类的反应时间一般处于毫秒级范围内。考虑到游戏的循环更新其刷新速率至少在50～60次/秒，因而可适当地降低各个游戏对象每秒内的逻辑执行次数，进而获取恰当的行为以及反馈状态。

对象的移动

对象的移动处理步骤负责更新其物理仿真状态，进而调整游戏对象的位置和/或方位，该步骤依然包含碰撞检测以及分解操作。多数游戏均会使用简单的技术获取角色的运动状态，以使其性能开销减至可控范围内。即使如此，高效地解决碰撞检测问题依然是难点之一。当模拟真实世界中的各种行为时（作用于游戏对象上的力学模型便是一例，如重力），基于数学计算的物理仿真常常过于复杂并会产生大量的计算开销，且伴随有碰撞检测以及分解操作等内容。但一切努力都是值得的：逼真的视觉效果、对象交互过程中的写实行为以及移动、碰撞时的可控操作均可视为上述方案的潜在优势。随着处理器性能的不断提高，一切皆有可能。这里，需要注意的是，针对于物理属性的仿真计算，物理模拟不应与游戏的可玩性背道而驰，亦即，真实感与娱乐性之间并无必然联系。

消息机制

消息机制常用于游戏对象以及游戏管理系统之间的通信，以表示发生了较为重要的游戏事件并需要修改游戏对象或游戏自身的状态。例如，游戏角色即将涉入某片水域（因而需要调整其移动属性或相机行为）、激活新游戏对象、开启一扇门、敌方角色处于死亡状态、数据资源加载完毕等。上述消息常通过脚本系统加以定义和操控。通过该方式，游戏设计者可规范并控制对象的行为，进而塑造出适宜的游戏元素，如谜题内容、播放视频等。一般情况下，相关消息可直接通过游戏代码或对象逻辑加以生成，两个对象间的碰撞检测便是一例。

技术要点

消息机制可有效地维持当前游戏状态，直至某一特定的时间段结束或当前消息与其他游戏事件同步为止。鉴于某些操作中的游戏对象将修改自身的游戏逻辑，因而这一类对象的状态变化需要在其“思考”步骤中进行调整；而某些对象则可适当地延迟至相关游戏对象逻辑执行完毕为止（在当次循环更新中，可规范全部对象间的一致性行为）。从理论上讲，各游戏对象应分别执行其自身逻辑并保持对象间的同步移动。但在实际执行过程中，常会出现某种隐式的操作顺序（有时则体现为显式操作顺序）。因此，游戏对象逻辑执行顺序将在消息发送队列中定义一个默认操作序列。除此之外，还可针对各消息设置相应的优先值。某些游戏引擎并不会阶段性地发送游戏消息，相反，待各对象的游戏逻辑全部执行完毕后，引擎将直接考察未处理的消息。更多与消息机制的相关内容可参考第6章。

需要注意的是，消息处理步骤并非总是出现于循环更新结构中。在某些情况下，一旦消息发送完毕，即可对其加以处理（通常位于“思考”步骤中）；或者延迟至全部游戏逻辑均执行完毕后，此时，全部未经处理的消息发送完毕。

针对游戏对象的逻辑执行序列，消息机制提供了有效的处理方案。对象在“思考”步骤中的游戏逻辑执行顺序并非总是显式地加以定义，实际上，该操作常根据游戏对象的创建时间加以设置。另外，消息机制还可有效地消除不同游戏系统之间的耦合关系；简化演示回放时游戏事件的记录操作抑或多玩家游戏中的远程复制操作；或者仅出于调试目的进行相应的简化操作。

相机

该步骤包含了游戏中的大部分相机逻辑，各个处于激活状态的相机均执行其逻辑内容，进而更新相机的位置和方向——分别由其行为和特定属性加以定义。当前相机的行为变化，抑或激活不同类型的相机都需要在当前场景渲染之前完成。相机操作将包含如下内容：

- ☑ 移除处于非激活状态的相机。
- ☑ 必要时，通过游戏状态的改变或脚本选项激活相机。
- ☑ 根据脚本内容或游戏逻辑，确定是否对渲染场景世界的当前相机进行调整。
- ☑ 针对当前处于激活状态的相机，执行其自身的逻辑内容，包括相机位置和方向。
- ☑ 针对主相机，对变化内容进行初始化，必要时还将包含插值的开始操作和结束操作。
- ☑ 基于主相机位置和方向，更新当前玩家的控制参考帧（用于描述玩家控制与玩家角色移动之间的关系，具体内容可参考第 2 章）。
- ☑ 构建用于渲染游戏场景视口的变换矩阵，其中还包括相关特效视觉内容，如抖动相机、视域、调整宽高比等。

构建必要的数据信息并创建一个或多个游戏场景视口可视为相机逻辑的最终目标。除此之外，相机还应向其他游戏对象提供相应的数据，这一类对象可能会执行与当前激活相机位置与方向相关的逻辑内容。需要注意的是，即使操作中的相机并未直接用于生成场景渲染视口，依然需要执行那个相应的相机逻辑，如两个活动相机间的插值计算。

相机后处理

待相机逻辑执行完毕后，通常需要对游戏元素进行更新，这一类元素往往依赖于相机系统的状态变化，例如第一人称游戏中玩家的武器系统。在这一类游戏中，武器系统的定位与方向常与玩家角色的视口相关，即依赖于第一人称相机的位置与方向。针对于此，相机的转换逻辑应在该相机的更新操作完毕后方可执行，进而可消除潜在的延迟问题（即位置/方位冲突）。

渲染器

作为更新内循环结构中的最后一步，渲染操作将生成与当前游戏状态相符的场景视口，或者向硬件发送信息以执行渲染操作。针对于此，渲染器需在全部游戏、相机逻辑操作完毕后方可执行。即使如此，创建于显示设备上的视口在下一次循环过程中依然可见。渲染步骤还可能涉及多个游戏场景的视口更新，一种情况是：渲染操作与视口无关，这常出现于多玩家游戏中。针对于与某一相机关联的游戏场景显示设备，各个视口分别体现了其中的部分内容。当然，视口间也可能出现交叠状况（例如

画中画效果，赛车游戏中的后视镜便是一例），因而需要对其渲染操作进行显式的排序。

相机视见内容的渲染操作并非总是输出至显示视口中（无论全屏与否），在某些情况下，还可将视见内容渲染至纹理中，这也意味着，针对于不同角度的当前场景世界视见内容，其生成过程均通过某一相机视口完成。例如，当玩家角色通过游戏场景内部中的显示设备（如显示器）对游戏场景进行观察时，便会使用到该技术。该显示器上的视见内容由渲染后的纹理构成，进而可在其视口内创建一类基于当前游戏场景的实时效果。该技术的计算量较大且需要遍历两个场景层次结构以确定最终的场景渲染内容（此处，场景内容彼此不同）。因此，读者应谨慎使用该项技术。关于场景构建的具体方案，可参考第2章中的相关内容。

常见的性能问题

游戏程序在诸多方面均对性能问题提出了较高的要求，包括图形解决方案以及构成场景世界的、数量众多且兼具一定智能的游戏对象，无论这一类对象是否与玩家存在着直接交互。在有限的资源环境下，针对良好的游戏可行度、玩家的完美体验以及实时条件下的实现方案，程序员以及相关设计人员应对此加以权衡。

在实时游戏程序中，常见的约束条件包括以下内容：

- ☑ 内存空间限制。
- ☑ 处理器性能问题。
- ☑ 图形处理器处理性能。
- ☑ 显示设备的分辨率。
- ☑ 控制器功能。
- ☑ 网络延迟以及带宽问题。

上述问题详细讨论的则超出了本书的叙述范围。通常情况下，简单的解决方案往往更加高效（当然，凡事皆有例外）。考虑到游戏环境测试以及漫游系统，相机系统对性能的要求可能会变得极为苛刻。

针对相机设计方案，正如Occam的剃刀理论所述，简单的相机设计方案往往更加高效。从美学角度上讲，简单的相机移动以及旋转操作便于观察者对问题的理解。然而，简单并不意味着易于实现。针对相机的场景描述以及游戏需求，二者之间的关系不应过于复杂，如角色的控制操作。除此之外，简单的相机设计方案通常不会包含针对特殊情况的处理过程，进而可加速相机的构建、测试以及调试操作。

相机系统

相机系统负责管理各种与相机相关的游戏特征，包括视频序列的播放以及实时游戏操控。根据特定的游戏架构，相机系统还将涉及场景的绘制步骤（即渲染操作）。简单而言，相机系统定义了各种环境下场景相对于玩家的表现方式。

另外，相机系统还应与其他游戏系统协同工作，并兼具数据信息的供给以及修正功能，其中包括

场景的渲染、实现控制器输入数据与玩家动作之间的映射操作等。同时，相机系统还应提供相关机制以供设计者根据游戏状态和玩家需求动态地修改相机的行为，这一类行为变化常源于玩家角色状态的变化（例如，角色的尺寸或移动特征产生了较为明显的变化）、环境变化（例如开放空间至狭窄、封闭空间之间的转换）等。需要注意的是，相机行为的动态变化依然涉及相机系统对优先权的管理。

技术要点

游戏刻画风格及其行为的多样化也使得相机系统对性能提出了不同的要求。例如，考虑到有限的游戏环境测试，第一人称相机往往对性能问题要求不高；而第三人称相机在遍历复杂的场景环境以及与其他游戏对象和当前场景环境实时碰撞检测时，将会大大增加处理器的计算量。本书第 3 部分将对如何减少这一类计算开销进行深入的讨论。针对于相机而言，期望的性能指标一直以来即是一个争论的话题且隶属于游戏设计需求范围。根据相关经验，相机逻辑不应超出处理器时间的 5%。另外，可将相机系统视为一个较为复杂的游戏对象，类似于执行特定人工智能的其他游戏对象（如 Boss），并将其性能开销视为一类近似标准。由于相机系统在游戏全部执行过程中均处于激活状态，因而应对其性能问题加以仔细考量。

综上所述，相机系统的职能包括以下内容：

- ☑ 管理所有处于激活状态的相机，确保其逻辑内容得到正确的更新。
- ☑ 根据脚本系统或调整游戏状态，以添加或移除相机。
- ☑ 考虑到相机行为或属性将随游戏状态的变化而发生改变，相机系统应对此进行相应的调整。
- ☑ 针对文件控制操作提供相应的参考帧。
- ☑ 处理非交互型视频播放序列。
- ☑ 针对相机显示属性，提供相应的控制操作进而对最终的渲染效果进行调整，其中包括视域、高宽比等。
- ☑ 通过图形用户接口管理并更新视口数据。
- ☑ 收集渲染引擎所需要的视锥体信息以正确地显示当前场景。
- ☑ 提供相应的调试选项，其中包括相机状态数据、针对独立可调相机的用户控制项，以及从不同观察角度获取的事件回放序列。

本书将重点讨论相机系统的设计和实现，相关内容将在后续章节中加以详细的阐述。

控制操作

关于游戏控制器，其使用方式决定了玩家角色的移动和行为方式，因此应得到足够的重视。作为玩家与游戏之间的主要交互方式之一，控制系统应向玩家提供一致、有效的接口。控制器硬件间的功能以及物理布局存在着较大的差异，这将大大增加控制器应用方案的选择难度。

玩家和相机之间的空间关系将严重影响玩家对角色的控制感，因而玩家的控制操作与相机系统间存在着紧密的联系。当然，这还将取决于游戏的表现方式以及风格。例如，对于第三人称视角游戏，当相机进行重定位时，相关控制操作则比较困难；而第一人称游戏常需要制订相应的控制方案，以使玩家的移动与相机的方位得以同步执行（即自由视角）。

一类较为常见的游戏技能组合了相机方位变化以及玩家的移动，即移动攻击，该技能以圆弧方式

围绕敌方角色移动且始终朝向该角色（玩家可保持射击状态或始终位于敌方角色前方）。其中，可通过侧面移动并辅以反向旋转实现上述运动方式，如图 1.3 所示。另外，还可调整上述操作之间的比例成分，进而修改移动的圆弧半径。上述内容可视为较为高级的操作技能，要求玩家具有较好的操控能力。同时，还可针对游戏引擎所定义的玩家角色（以及相机）的反馈程度实施进一步调整。

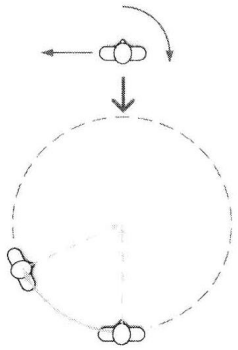


图 1.3　移动攻击使得玩家角色可相对于场景中的另一个对象或位置进行移动

类似地，诸多第三人称游戏针对玩家提供了上述功能，并动态地改变相机的位置和/或方向且独立于玩家角色。玩家操控和角色移动之间的关系常称作控制参考帧，第 2 章将对此加以讨论。

物理控制器硬件也将影响到相机风格的选择。例如，在基于计算机的角色扮演游戏中，鼠标和键盘提供了接口选择方案，而这一类设备通常在游戏机上无法得到。因此，与游戏机玩家相比，无论是玩家角色控制还是相机定位，计算机玩家均对此有不同的感受。控制器硬件的深入剖析则超出了本书的讨论范围，但设计者应对玩家控制操作以及相机用途有一个较为清晰的认识。

脚本设计

针对多数游戏而言，游戏事件或游戏状态变化的动态反馈能力其重要性不言而喻。类似地，当发生相应的游戏事件时，应确保游戏状态以一种统一、预期的方式变化，例如，当进入某一区域或操控某一对象时玩家所出现的动作反馈。脚本系统对此进行了详细的描述，该术语源于电影脚本这一概念。众所周知，电影脚本规定了事件序列、拍摄技术以及影片中的对白。同样，游戏脚本机制也可用于控制、定义相关事件和行为，进而向玩家展示某种游戏体验，其中也会包含视频播放序列的构建过程。考虑到交互性特征，与传统的影视剧本相比，游戏脚本机制则要复杂得多，且需要顾及玩家行为所包含的变化特性，这一点远非简单、直接的影视作品可比。

脚本机制不仅限于向玩家展示游戏情节，还可改变玩家与游戏之间的交互方式，包括（且不限于）物理控制器与玩家角色之间的关系。例如，游戏过程中可令特定的玩家控制失效。玩家控制常依赖于相机系统，这也是本书读者所关心的重点内容之一。除此之外，游戏还应根据玩家动作呈现应有的方式，并可在特定的复杂环境中展示令人满意的视觉效果——这种调整能力均可视为相机系统的重要特性。

最后，无论是游戏设计还是其他特定领域，相机系统还应包含相关功能以满足特定的移动、定位、定向等控制操作。第 5 章和第 6 章将分别对脚本机制进行深入的讨论。

针对于设计者而言，游戏事件或相机脚本机制可采用文本（或类程序语言）加以编写，但这并非是唯一解决方案，具体内容将在后续章节中加以阐述。需要注意的是，脚本内容切忌过于复杂：对于设计者而言，应易于理解；对于使用者而言，应相对直观。这里，应避免脚本系统产生特征蠕变，亦即，期望特征不断增长，导致开发周期远远超出规定时间。有时，某些较为复杂的问题更适于在程序环境中加以解决，且可方便地访问内部游戏系统（并拥有强大的调试能力）——这对于脚本系统确实勉为其难。

本章小结

本章考察了典型实时应用程序的基本特征，以及相机系统的诸多功能。这一类程序的动作管理序列称为循环更新结构，并可视为一系列相对独立的操作步骤，其顺序和体现方式不一而同，在某些情况下，同一程序内甚至会出现不同的更新操作。相关操作步骤多出现于视频游戏中，对应示例在本章开始处亦曾有所提及。

本章所阐述的相关原理同样适用于其他形式的交互程序，但可能会省略某些步骤。需要强调的是，相机逻辑通常在循环更新结构后期执行，此时，其他游戏对象中的逻辑内容应执行完毕。另外，与当前相机相关的相机后处理操作同样不可或缺。

本章还列举了相机系统的诸多功能，需要注意的是，功能需求往往根据程序特质以及执行环境的不同而发生变化。同时，本章还讨论了用户控制操作以及游戏事件脚本机制并进一步阐明：相机系统逻辑并非独立存在且往往依赖于游戏中的其他重要属性。

第 2 章　相机基础知识

本章将讨论与实时相机相关的基础知识，包括游戏相机的基础特征、与真实相机之间的比较以及相机的常见属性。

如何定义实时相机？玩家通过游戏场景显示其自身的存在，该过程的难点是什么？这里，读者需要理解游戏相机操控的重要性，考察下列情形：通过当前相机，玩家仅可观察到场景的部分内容。如果相机的位置和方向阻碍了玩家的视野或移动方向，亦即，如果相机在移动或重定位过程中出现抖动等不稳定状况，这通常会令玩家难以接受。另外，读者还应牢记显示操作的本质特征：将图像投影至二维屏幕平面以实现虚拟场景的二维或三维成像。这与摄影以及影视技术有着异曲同工之处，尽管当前视见内容实时生成且具有不同的美学要求。

游戏相机的定位以及方向将对玩家的控制感产生较大的影响，因而在游戏中具有举足轻重的地位。优秀的相机系统可较好地融入游戏场景中且对玩家而言是透明的，并对游戏体验起着正面作用。

真实的相机系统

真实相机及其属性构成了游戏相机的基础理论。本书大部分内容主要讨论持续运动的场景对象的显示方案，而非静态相片。相关示例表明，上述解决方案与静态图像技术并不冲突。

大多数读者对影视制作以及生活中的相机设备并不陌生（无论读者是否意识到这一点），游戏的目标之一便是尽可能地还原这一类真实的视觉效果。另外，影响拍摄效果的因素通常包含以下几点：

- ☑ 镜头类型（例如，变形透镜、放大透镜以及原镜头等）。
- ☑ 图像捕捉方案（基于胶片的光学成像抑或数码成像）。
- ☑ 胶片类型（胶片尺寸，例如，35 毫米胶片和 70 毫米胶片，胶片粒度，感光度等）。
- ☑ 捕捉频率（电影胶片每秒播放 24 帧，有时可高达 96 帧/秒）。
- ☑ 胶片曝光（根据快门速度，可能会出现过度曝光或曝光不足）。
- ☑ 投影方案（数字图像投影或传统的胶片投影）。
- ☑ 宽高比（投影设备的尺寸比例，即宽度值除以高度值）。

在交互式实时应用程序中，上述特征将受到多方面的限制，其中包括处理器性能以及渲染能力。随着图形处理器以及中央处理器的不断发展，越来越多的素材可实现相应的模拟操作。因此，针对展示相关特效所担心的性能问题，开发人员可对游戏引擎中的非交互式视频播放序列（并非一系列的图像序列）实施更多的操控。

实时着色语言以及其他高级渲染语言的出现大大提高了基于真实相机场景的模拟效果。然而，相机设备捕捉图像时的真实物理行为与经计算得到的游戏相机视见内容二者之间存在着较大的差异。这里，计算机生成的、经过预渲染的视频序列通常会通过相机镜头以及胶片的吸收作用重新生成光线路

径（即光线跟踪）。实时相机的渲染过程往往会采用相应的数学技巧将游戏场景内容投影至二维平面上以形成最终的渲染图像。

游戏相机可以采用物理模型（或近似模型）计算对象表面所产生的大气反射或折射现象，这一类技术方案可用于生成人工场景并可模拟人眼中的世界。实时应用程序（如游戏）采用各种近似解决方案以显示虚拟世界景观，并可达到较高的仿真程度。随着渲染性能的不断提高，物理解决方案很可能会替代多边形显示方案，相关内容可参考[Pharr04]和[Akenine-Möller02]。

另一方面，实时相机亦包含了真实相机设备所不具有的诸多优势，尽管某些内容并非是所期望的相机行为。真实相机设备根据其不同用途已建立起一套相对完善的规范，且随着时间的推移已为广大消费者所接受，即所谓的摄影技术。相关技术描述了相机的移动方式、视野范围内拍摄主题的定位方式以及不同场景间的切换方案。多数情况下，违反上述拍摄要求（无论有意或无意）将会产生令人难以满意的效果。

当尝试重现场景以及拍摄行为时，尤其是实现所谓的拍摄序列时，应遵循上述规范。在游戏体验过程中，情况则稍有不同。然而，上述规则依然存在较大的参考价值，甚至会产生令人意想不到的结果。

在当今主流电影工业中，计算机视效以及数字捕捉技术的应用使得游戏相机与真实相机设备之间的差异迅速减小。当在场景与计算机视效之间实施无缝接合时，运动轨迹数据可对相机的运动方式实现精准的复制。其中，可对光线以及其他场景数据实施有效的捕捉，以进一步完善真实场景与特效间的差异。针对整部影片而言，后处理操作通常不可或缺，进而可调整光照条件、移除胶片中的瑕疵、置入数码数据等。体育赛事转播中常会看到此类情景，通过相机的移位进而模拟视频游戏中的画面效果（例如，可将摄像机安置至飞艇上或悬挂于足球场上方线缆上）。

类似地，实时应用程序还可应用于预视频技术中，亦即，在提交渲染任务之前，通过相关拍摄素材或对生成后的视频序列进行预览进而实施详细的前期规划。

用途和属性

第 3 章将讨论基本的摄影技术及其与实时相机应用之间的关系。读者可参考[Hawkins05]和[Arijon91]等相关资料以获取更为深入的内容。这里，相机设备（尤其是电影摄像机）还包含下列特征：

- ☑ 缺乏交互性。在电影拍摄过程中，各场景以同一方式加以展现且观众无法对此作出任何改变。这也使得某些特定的场景处理方案可掩饰当前技术中的某些不足之处，抑或进一步放大相关缺陷。当然，观众一般无权操控摄像机，因此自然不会生成与电影导演相悖的画面。
- ☑ 预先确定拍摄视点。摄像机的摆位应符合相关美学标准，这对于观者的感受以及特定事件的展现至关重要，并可用于强调宏大的视觉场景。针此一系列的拍摄手法，相信观众不会感到陌生。当然，摄像机的特定置位还可达到欺骗观众的效果并使其对某种错觉信以为真（例如强迫透视，具体内容将在第 3 章加以讨论）。
- ☑ 相机镜头。相机镜头可用于增强影视元素的拍摄效果，还可对场景画面产生扭曲效果，例如拍摄对象间的聚焦变化，即景深效果。需要注意的是，这一类视觉效果亦可应用于后期操作中。
- ☑ 场景变化。交叉淡化、换景、过渡效果以及其他较为常见的“标准”拍摄技术常出现于电影场景间的变化过程中。由于上述效果通常在视频编辑阶段进行操作，因而可充分表达某种特

定用途或期望的感观状态（如强调场景中的某一特定人物）。多数效果均可通过计算机加以生成，这将大大降低真实拍摄设备与虚拟交互相机之间的差异性（但性能问题依然使得实时应用程序无法实现某些视觉效果）。

- ☑ 胶片曝光。根据各胶片帧的曝光时间，摄像机可实现一类运动模糊效果。在光圈开启时，摄像机将对当前场景内容加以捕捉，因而胶片的曝光时间可使对象的位置产生变化。同时，鉴于运动方向的各异性，该过程将使图像处于模糊状态，进而强调物体的运动特征。
- ☑ 后处理操作。除去常规编辑操作之外，该过程通常包含着色过程以及色彩修正操作（例如场景间的过渡变换）。考虑到这一类效果往往在胶片投入使用前进行加工，因而需要投入较多的人力和时间。另外，某些后处理操作还需要使用到基于数字图像捕捉的原始胶片。

电影投影

针对于最终图像的显示问题，电影和游戏具有相同的目标。在影院的投影设备中，连续胶片帧的物理显示过程存在着某些限制条件，当采用数字技术时，它们并未得以完全体现（无论是采用数字格式存储的电影或游戏）。

但物理胶片投影至屏幕上时，光线通过镜头、胶片进行投影并聚焦于某一平面之上（即电影屏幕）。由于各帧均需（连续）完美显示，因而需关闭其他光源以防止胶片连续播放时产生错误的投影效果。当该处理过程以较慢的速度执行时，由于人眼可察觉到帧间的过渡图像（通常显示为黑色），因而全部连续画面将产生较为明显的闪烁效果。在实际应用过程中，当更新速率达到 24 帧/秒时，即可对播放序列进行有效的模拟；当更新速率达到 48 帧/秒时，即可消除显示过程中潜在的闪烁现象。

随着显示技术的不断发展，电影的投影方式也得到了较大的改善。目前，整部电影可采用数字格式加以存储，其分辨率甚至可超出胶片格式。因此，数字投影已成为一种切实可行的解决方案。另外，数字投影技术还拥有与视屏游戏技术相同的特征。除去数字媒体常见的各种优点之外（例如超长寿命以及稳定的画质），针对三维视觉效果的表达，这一类投影技术包含了诸多潜在优势，且观察者无须佩戴特殊的光学视镜。另外，上述处理方案还可完全消除胶片投影中固有的闪烁现象（由于相邻帧之间可实现同步投影，因而可使用顺序扫描显示技术）。鉴于显示影像通常尺寸较大，因而数字胶片投影技术往往需要较高的分辨率以及刷新率。因此，与传统的胶片投影技术相比，数字投影技术将带来全新的视觉体验。在编写本书时，数字投影技术依然成本较高且应用范围有限。

游戏相机

游戏相机可视为一类抽象结构，并向玩家展示游戏场景画面且兼具真实相机以及投影仪的功效。正如读者所料，游戏相机根据游戏场景中的视点位置和方向定义了多个属性。更为重要的是，显示风格反映了当前游戏场景在显示设备上的投影（或渲染）方式。除此之外，其他渲染属性还包括视域、宽高比等。

同时，读者还可进一步考察相机与玩家移动以及其他游戏元素之间的交互方式，亦即相机类型或

相机行为。除去场景中相机的真实运动以及重定向之外，相机行为方式将直接影响其期望位置以及方向。针对于玩家，在定义游戏场景的显示方式时，上述属性通常不可或缺。

针对于游戏应用，存在多种特征可区分游戏中的相机行为，最为简单的一种判别方式是游戏相机激活时是否触发了游戏行为。亦即，可通过相机类型（电影相机抑或交互式相机）对此加以区分。

游戏内的电影相机与真实相机相比，二者之间拥有诸多共同点，其中最为重要的是，玩家或其他游戏对象不会对相机的表现方式产生任何影响。另外，真实相机所展现的行为方式通常可在游戏版本中加以复制，并可在相应的拍摄规范之上添加其他操作。例如，真实相机可能无法实现游戏相机的移动以及行为方式，电影摄像机每次均采用同一方式显示当前场景，且仅在必要时根据游戏状态对象场景元素加以修改（如角色的外观）。通常情况下，场景组成元素将保持不变。

当使用电影相机时，某些辅助画面的状态常会发生改变，进而提示玩家将不再拥有对角色人物的控制权。有时，相关变化还体现于渲染视口宽高比（相对于当前显示设备的宽高比），并以此强调电影摄像机的非交互特征。交互式相机与电影相机之间的主要区别在于：交互式相机可调整其行为并对实时游戏事件加以反馈——无论是玩家动作抑或是其他游戏元素。但是，交互式相机所面临的较为困难的任务是：玩家（抑或其他游戏对象）所执行的动作变幻无穷，且需要生成对应的模拟操作。因此，交互式相机需要针对游戏的各种变化做出反应，并在任意时刻显示正确的场景画面。

交互式相机和电影相机均包含诸多特征，但本书将集中探讨交互式相机的设计和实现方式。如果不做特殊说明，后续内容将仅涉及交互式相机。如前所述，游戏相机包含诸多难点且难以复制真实相机所拥有的行为模式，其差别在于：游戏相机固有的交互特征使其需要对游戏状态加以动态调整，最终生成正确的游戏画面。在交互序列中，大多数胶片技术无法直接应用于游戏中，这多少令人感到遗憾。

下列内容体现了游戏相机所涉及的固有特征：

- ☑ 游戏相机应兼具真实相机以及投影仪功能。
- ☑ 游戏相机无须过多的物理特征，但应满足空间内的移动操作，其中包括基于玩家视点的障碍物穿越行为、即时重定位等变换操作。
- ☑ 相机行为或属性将根据游戏需求动态改变。
- ☑ 根据相关游戏需求以及美学要求，游戏场景将针对现实设备采用不同的投影方案，进而以彼此不同的方式展现同一场景世界。
- ☑ 光照以及渲染滤镜应根据不同的游戏需求以及设计理念加以调整。
- ☑ 相机与场景之间包含多种过渡变换。
- ☑ 场景渲染可用于特殊的视觉效果中，并在响应游戏事件时增强或夸大场景元素的表现力。

基本技术

下面将首先讨论某些基本技术并对游戏相机及其相关事件加以定义。需要说明的是，当涉及对应的电影术语时，游戏相机则包含不同的含义并需要重新加以定义。当描述相机系统时，常会谈论到第一人称相机和第三人称相机，但如何对其加以定义呢？这一类相机是否如其名称那样具有不同的含义？这里，存在若干明细规则针对实时相机加以描述，但某些技术并不具备明显的可描述特征，因此，这里将针对基本的相机系统功能加以定义，后续章节还将对此进行更为深入的讨论。

相机系统

相机系统涵盖了多种彼此不同的整合元素，使得游戏可对虚拟世界的表现方式加以控制。在第 1 章中曾提到，相机系统可视为游戏引擎的主要组件之一，具体内容如下：

- ☑ 应对处于活动状态下的游戏相机进行必要的管理，并在相机间执行必要的转换操作。
- ☑ 相机可视为包含特定行为的应用程序，包括相机的定位、移动以及方向。
- ☑ 根据游戏状态对相机行为的变化进行操控。
- ☑ 生成相关信息以供图形引擎使用，进而渲染游戏场景世界。
- ☑ 针对于玩家控制操作，相机应提供相应的参考帧可控操作。

游戏相机

关于游戏实体，其抽象的解释可描述为：该实体包含了足够的信息用以当前场景世界的位置和方向。游戏相机通常会包含相关的属性，进而确定场景内的期望位置和方向（例如实施跟踪的目标对象）。在某些情况下，游戏相机还应包含构建场景视野所需的相关元素，其中包括视域以及其他渲染信息。同时，这一类信息还将在其他结构中出现，例如视口。

表现风格

将游戏场景投影至显示设备上的相关方法可称作二维或三维相机系统，具体内容可参考“相机行为”小节。

相机行为

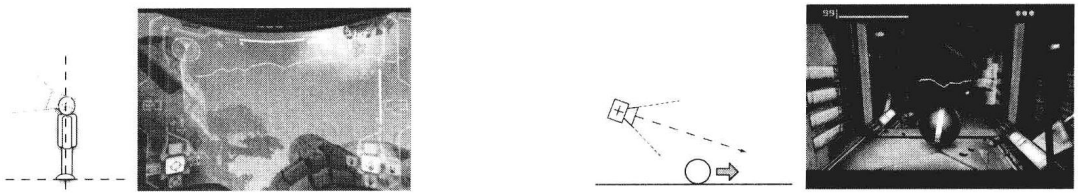
相机属性（如期望位置、视域、移动特征等）经适当整合后可用于描述游戏场景的视见内容。

第一人称相机和第三人称相机

第一人称相机和第三人称相机定义了不同种类的相机行为。在某些情况下，这一类行为与显示风格具有相同的含义。不仅仅是与玩家角色相关的位置（以及方向）数据，某些基于相机系统的处理方式也大同小异。

尽管如此，某些渲染方式仍然持有不同的处理方法。相对于第三人称相机，第一人称相机一般会提供一个包含某些限制条件的子集。图 2.1 显示了相机的位置及其关联视野。关于相机行为的更多内容，

读者可参考“视见生成”一节。



（a）第一人称相机的位置和视野　　（b）第三人称相机的位置和视野

图 2.1　相机的位置及其关联视野

电影相机和交互式相机

通过前述定义可知，游戏可视为一类交互式媒体，因此，本书将重点讨论交互式相机的设计与实现。亦即，相机的行为将根据游戏状态的动态变化而改变。另外，视频播放序列可能会出现于游戏中并展现真实的场景画面。

观察位置

相机的期望方向可通过多种不同方式加以定义，一类常见的方法是定义相对于目标对象的位置（即根据目标对象的方向和位置）并令当前相机重新定向。如图 2.2 所示，在第三人称相机中，观察位置可独立于目标对象。

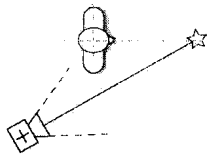


图 2.2　第三人称相机的观察位置与目标对象之间存在着显著的差异

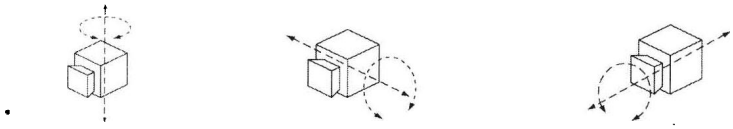
期望位置

针对相机的移动，期望位置定义了世界坐标系统中的位置数据，该位置的最终确定应根据具体情况加以分析，相关内容可参考第 7 章。

方向

在世界坐标系统中，相机方向确定了当前相机的朝向以及相机的视见空间，其中包括基于镜头的

场景画面和围绕视见方向上的旋转操作。这里，方向可通过多种方式加以确定，具体内容可参考第 7 章。其中，方向包括 3 个分量，即偏航角、仰角以及旋转角，如图 2.3 所示。



（a）偏航角　　（b）仰角　　（c）旋转角

图 2.3　偏航角、仰角以及旋转角

偏航角

这里，偏航角是指相机的水平方向，在世界坐标系统中，偏航角可通过围绕 Up 轴的旋转角度加以定义。

仰角

相机的垂直定位方向，即相机围绕其右向轴旋转，该角也称作倾角。

旋转角

相机围绕其前向轴旋转则称之为旋转角。

期望方向

相机的首选方位则根据其行为类型加以定义，通过适当的旋转操作，相机可平稳地过渡至该方向之上。有时，相机的方向与其期望方向可实现同步调整，因而无须使用旋转操作。

旋转操作

旋转操作是指一定时间范围内的角度变化，常通过角速度加以描述。

视见体

将相机和视口经过适当组合后，视锥体可形成游戏中的部分可见场景，具体内容稍后加以解释。

视口

视口可视为一类数据结构，对应于显示设备的可见部分，并用于显示基于相机的可见场景。视口

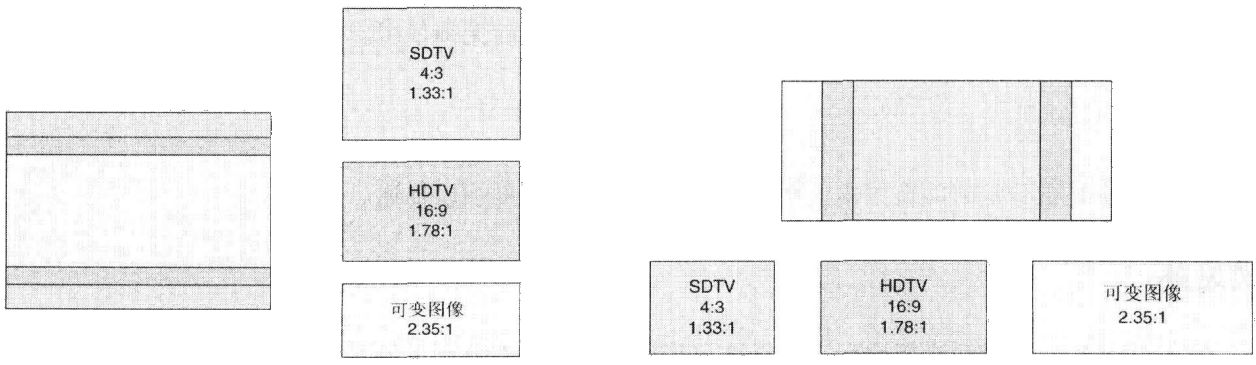
的形状和尺寸通常会发生变化，且在游戏运行时呈现为动态变化。在某些情况下，当前视口可能会与其他视口或图形元素发生重叠（例如，赛车游戏中的后视镜视口通常位于全局游戏视口之上），因而需要对渲染操作进行排序。另外，视口也可使用其他形状，而非总是矩形。在输出至渲染步骤之前，视口往往还需要其他辅助步骤，例如，可在前述图形绘制元素中使用某种屏蔽遮挡操作以形成当前视口。同时，多玩家游戏一般会针对每一位玩家提供独立视口（并可根据玩家的数量、玩家角色人物的尺寸等内容适当地调整视口尺寸以及位置）。

视域

通常情况下，视域（FOV）是指视锥体内顶部和下方界限之间的角度值，进而定义了游戏场景垂直方向上的投影可视范围。除此之外，FOV 还可定义为水平角度（常见于美工师所使用的渲染软件中）。针对于此，读者应确保该操作的一致性。水平 FOV 和垂直 FOV 一般会涉及设备的宽高比，第 10 章将对水平 FOV 以及垂直 FOV 之间的转换操作（以及反向转换操作）进行详细的分析。

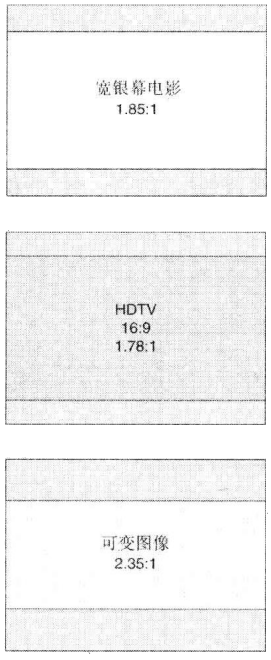
宽高比

宽高比定义为水平视口尺寸与垂直视口尺寸之间的比值。针对于标准电视设备（SDTV）的全屏视口，其值通常为 1.33（即宽高比为 4:3）；针对宽屏或高清电视（HDTV）设备，该值则为 1.78（即 16:9）；电影投影中的宽高比则不一而同，较为常见的两个数字是 1.85（即宽银幕电影）和 2.35（即可变形宽银幕）。基于 FOV 的宽高比定义了游戏场景中的可见部分。关于视频标准、宽高比以及其他相关内容，具体内容可参考[Poynton02]。在某些情况下，游戏场景渲染视野的宽高比无法与现实设备实现良好的匹配。例如，考虑到与电影投影间的某种类似性，宽屏宽高常用于表现缺乏交互特征的画面（不考虑物理设备的宽高比）。图 2.4 显示了多种宽高比之间的比较结果及其与标准显示设备之间的关联性。

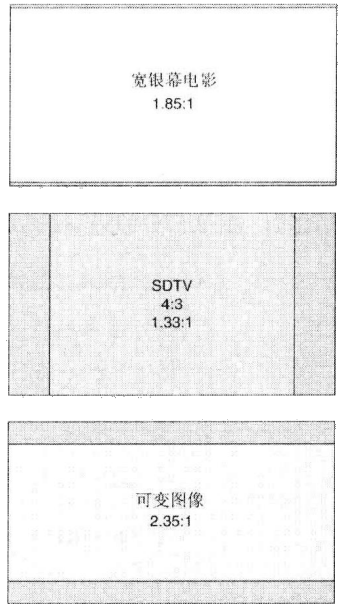


（a）常见显示设备宽高比的相对尺寸

图 2.4　多种宽高比之间的比较结果及其与标准显示设备之间的关联性



（b）16:9 显示时的宽高比比较



（c）4:3 显示时的宽高比比较

图 2.4　多种宽高比之间的比较结果及其与标准显示设备之间的关联性（续）

刷新率和帧速率

刷新率或帧速率（有时也称为更新率）用于测试硬件显示设备每秒的执行次数，也可用于标识游戏逻辑的更新率。理想情况下，该值应保持一致且与图形细节的数量以及当前游戏的处理器需求无关。亦即，刷新率主要取决于游戏本身。通常，更新操作应与显示设备的物理刷新率保持同步。在现代游戏系统中亦是如此，其中游戏逻辑并行于图形更新操作。若图形更新操作超出了固定频率值，某些游戏则会关闭同步功能。当然，在显示设备执行更新操作时，这将产生波纹效果并使得视觉画面变得扭曲。

针对于玩家控制操作以及平滑的相机更新操作，一致的帧速率是十分必要的，帧速率的改变往往会令玩家处于混乱中并使角色的运动效果大打折扣。

当相机状态变化根据距前次更新操作的流逝时间加以计算时（即 Δt），游戏相机可不受帧速率的约束。

安全区和安全帧

安全区是指显示设备四边框内的区域，物理显示边框可能会对该区域产生某种弱化作用。例如，显示设备的曲率较大，抑或显示部分与真实屏幕相互交叠。然而，现代显示设备多采用纯平屏幕且配

以较窄的边框，因而上述问题不复存在。即使如此，由于该区域表示为视见外围，应尽量避免将较为重要的状态信息置于此处。另外，该区域的尺寸还取决于物理显示分辨率。针对 SDTV 显示设备，32 像素可视为较为适宜的尺寸；而对于 HDTV 显示设备，其边框应适当地加倍（根据物理像素）以配合较高的分辨率。

相关经验表明，可将显示设备分辨率的 10%～15%留于边框区域。总之，安全区的目的是确保重要信息针对玩家可见，相关尺寸可根据现实信息的特征加以调整。针对于此，位于显示信息内部的显示区域则称作安全帧，图 2.5 分别显示了安全帧以及安全区。

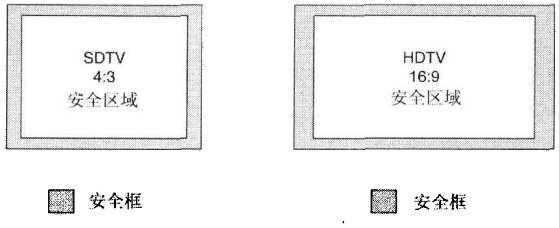


图 2.5　SDTV 和 HDTV 显示设备的安全区和安全帧

刷新率和运动模糊

随着时间的推移，低刷新率导致的模糊现象将不再视为问题，但在编写本书时，该问题依然需要得到足够的重视。在这一点上，刷新率所导致的模糊现象与运动模糊存在着较大的差异。在早期的 LCD 显示器上，刷新率所导致的模糊现象常会产生令人意想不到的结果，其中，显示媒介无法根据画面的变化快速地显示计算机的输出内容。最终，显示结果往往缺乏应有的精确度，从而导致游戏的视觉效果大打折扣。同时，这一类效果常会出现于显示设备的全部画面中。

运动模糊最初源于胶片有限的曝光时间。快速移动的物体常会占据较大的区域或在尾部留有模糊的拖影。在胶片通过相机取景器曝光的一段时间内，快速移动的物体将在同一胶片帧中留下多个位置。图 2.6 显示了运动模糊效果。



图 2.6　当物体移动快于胶片的曝光时间时，则会产生运动模糊效果

在某些情况下，运动模糊可掩盖渲染技术的不足，例如，该技术可用于渲染电视帧中交替变化的扫描线。有时，图形硬件在生成其输出内容时会留有前次扫描线信息，进而在创建计算机图像时会消除某些缺陷图案。类似的效果还出现于倍线器这一类硬件中，与 HDTV 显示设备相比，当某些低端或标准电视设备的分辨率较低时，该方案可进一步丰富画面的数据信息。

显示撕裂

在物理显示设备更新过程中，当渲染内存缓冲区中的内容被复制至显示内存中时，抑或该内容被切换至活动的显示缓冲区中时，可能会出现显示撕裂现象。这里，最新渲染数据与仍然处于显示状态的上一帧数据之间存在着显著的差异，这取决于显示设备的图像生成方式且最终结果并不确定。

玩家角色人物

在不同的游戏类型中，玩家角色的表现方式也不尽相同。在本书中，玩家角色通常定义为人物、车辆、人类、生物以及游戏内其他玩家可控对象。通常情况下，角色之间的差异需要采用不同的相机方案加以处理。同时，相机的期望位置以及视点位置还应借鉴玩家角色的位置和/或方向及其移动特征。在某些游戏中，玩家角色这一概念并不明显且不会在游戏中加以明确体现（如第一人称游戏）。

目标对象

目标对象可视为一类游戏对象且用于指明相机的视点位置。在某些情况下，相机的方向将参照该对象，尽管在指向目标对象的过程中存在延迟。另外，真实的视点位置往往与目标对象之间存在少许偏差（偏移），具体内容将在第 7 章中加以介绍。在多数情况下，目标对象可视为玩家角色人物。

技术要点

在现代游戏应用程序中，由于缓冲区之间的切换操作（必要时还将包含复制操作）与垂直消隐（VBL）信号同步执行，因而显示画面的撕裂效果鲜有出现。在显示硬件未读取显存时（该过程通常较短暂），往往会执行上述同步操作。为了有效地消除图形处理器或中央处理器的延迟现象，可在渲染时在多个内存缓冲区之间进行切换操作。根据存储缓冲区的数量，上述切换操作可相应地设置为双缓冲或三缓冲，这种显存之间的瞬时切换可有效地消除撕裂现象。然而，内存消耗可能或产生某些负面影响，在某些受限系统中，内存环境难以支持上述切换操作，在这种情况下，撕裂效果难以避免。最终，画面的显示效果也将大打折扣。如果显存内容需要复制至输出缓冲区内并由显示设备加以读取，则复制操作自身可与 VBL 信号同步。经过谨慎处理后，显示设备可在复制过程之后读取内存（即并行操作），进而可有效地消除撕裂效果。

上述过程与所采用的显示技术紧密相关，当渲染内存与全显示分辨率不一致时尤是如此，这将涉及交错渲染技术，并交替改变扫描线的输出内容，进而降低内存的占有率以及 CPU 需求，并充分发挥某些显示设备的优势（如 CRT 显示器）。

插值计算

简而言之，针对一段时间内相机属性的平滑改变，插值计算采用了一个算法集并用于调整相机间的不同行为。通常情况下，除了可控制 FOV 或其他属性变化之外，插值计算将以平滑的方式修改相机

的位置以及方向，第 10 章将对此加以详细的讨论。

投影

投影操作可将游戏场景映射至当前显示设备之上，并采用相应的投影变换加以实现，进而模拟人眼所产生的立体视觉。图 2.8 显示了典型的透视投影方案。平行变换则是一类较为常见的选择方案，其中，在投影过程中，平行线依然保持平行。除此之外，正交投影、轴测投影以及正等轴测投影也十分常见，具体内容可参考第 4 章。

视差

视差源于天文学术语，当以不同的视角对物体进行观察时，该物体貌似发生移动。在游戏中，根据场景渲染元素与相机位置之间的距离，视差常用于描述其相对运动，例如 2D 滚屏游戏中的图层、地平线的模拟以及天空盒（亦即，使用少量几何体对象模拟远距离物体的一种渲染技巧）。

相机转换

处于活动状态的游戏相机可能独立存在于场景中，其行为与其他游戏对象具有较为显著的差异，因此，游戏相机系统需对其进行调整（如移动或方向等属性）。针对于属性各异的相机行为，其调整方式称之为相机转换操作。例如，玩家可在第一人称相机和第三人称相机之间进行切换。

相机转换通常会涉及特定的处理操作，并确保该过程以一种平滑、适度的方式进行。除去瞬间中断这一特殊情况，建议应在某一时间段内对相机属性进行调整，并使得当前游戏得以正常运行。需要注意的是，某些转换操作可能会被后续的调整操作所打断。

常见的转换操作包括以下内容：

- ☑ 相机行为的调整（例如，相对于目标对象的距离或高度、速度等）。
- ☑ 插值计算（一段时间内位置和方向的调整，包括 FOV 等）。
- ☑ 相机的显式风格（例如第一/第三人称相机）。
- ☑ 跳接处理（需要对方向以及位置进行即时调整）。
- ☑ 清除操作（清除当前屏幕数据并通过图形手段对场景加以恢复）。
- ☑ 淡入/淡出效果（将当前场景数据褪色至某一颜色，一般为黑色或白色，并在两个不同场景间实现交叉渐变效果）。

相机约束条件

针对游戏相机的期望位置和方向，一类常见的方法是使用一系列的、基于分层的约束条件。其中，

相关限制条件以显式顺序作用于对应的期望属性之上（例如，相关限制条件作用于移动项之后，可继续实施碰撞约束条件）。针对当前各项属性，全部约束条件汇总后即定义了相机的行为。另外，根据游戏需求，必要时还可添加或消除相机约束。这将有助于单独考察相机的移动和方向。

移动约束条件

如果游戏相机的移动缺乏相应的约束条件，则很容易导致相机穿越几何体这一情形，且无法与快速移动的游戏角色实现有效的绑定，最终的显示效果将难以预料。因此，当确定有效的相机位置并充分显示游戏体验的各项特征细节时，对相机进行相应的限制是十分必要的。

移动约束条件包含下列内容：

- ☑ 2D 或 3D 距离。
- ☑ 与目标角色之间的相对位置。
- ☑ 相对于世界坐标系的位置。
- ☑ 路径（直线路点、样条曲线等）。
- ☑ 隐式表面。
- ☑ 加速限制。
- ☑ 速度限制。

关于移动约束条件，第 10 章还将对此进行更为深入的分析。

方向约束条件

方向约束条件常用于第一人称相机，尤其是基于全视角的玩家直接操控。这里，术语全视角用于描述玩家在游戏体验过程中的相机重定位能力。全视角约束条件包含相机的最大、最小倾斜角，还可能包含与参考帧相关的偏航角。通常情况下，人物中的偏航角移动可消除其上的偏航角约束（第一人称相机一般仅与角色人物的前景相关）。有时，当角色人物的方向保持固定时，观察视野可通过其视见方向加以控制。

全视角第三人称相机也包含了类似的约束条件。在某些情况下，用于跟踪玩家且具有固定位置的相机还可包含相应的约束条件以禁止垂直方向变化。

- ☑ 对偏航角加以控制以避免方向失控。
- ☑ 适当地添加阻力（软阻力或硬阻力）。
- ☑ 针对细微的变化，应设定相应的加速限制。
- ☑ 通过角色的速度影响方向变化。
- ☑ 限制旋转速度以保证正常的运动效果。
- ☑ 相对于目标对象，可将相机的观察位置限制在某一特定位置上，或限制其速度和/或加速度。通常情况下，观察位置的计算将根据玩家角色的位置、移动、速度以及行为发生改变。

相机的表现风格

下面将考察游戏场景的表示方式，由于相机将直接影响观察者的视觉观感，因而可将其表现风格视为游戏相机的重要特征。这里，表现风格通常定义为游戏场景的正交渲染（二维）或透视渲染（三维）。为了简单起见，上述表现方式常称作 2D 相机或 3D 相机，尽管这种归类方式并不完全准确。除此之外，还存在一类混合表现方式，其中，游戏场景或游戏对象均采用正交渲染（尽管视觉上呈现为三维渲染），这一类表现方式一般称作 2.5D 相机。下面将对此进行逐一分析。

正交表现方式（2D）

二维表现风格源于渲染过程中的投影方式，即平行正交投影。在正交投影过程中（如图 2.7 所示），平行线经投影后仍保持平行特征。尽管游戏场景及其内部对象的图形表现方式定义为 2D，但相机行为依然可表现为 3D 特征。

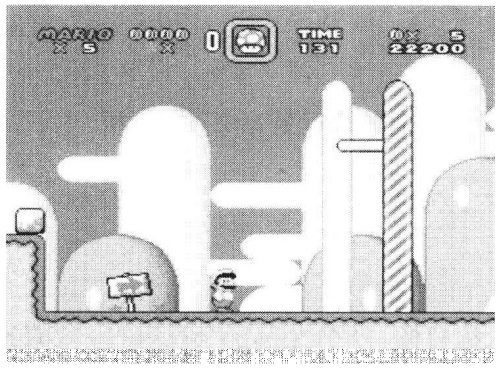


图 2.7　典型的正交相机表现方式。图像源自 Nintendo 公司 1991 年出品的著名游戏 Super Mario World

2D 相机系统包含以下特征：

- ☑ 游戏支持滚屏或背景图案保持静止。在某些情况下，还可通过分层机制提供场景的深度感——当采用视差时，该效果将十分明显。另外，背景图案常由较小的元素构成，例如贴图机制。其中，小尺寸贴图元素经适当组合后可形成较大的贴图并可采用对应的索引加以管理。同时，贴图机制还可视为一类标准尺寸的静态像素布局方案。例如，贴图元素可定义为 16×16 像素。
- ☑ 相机的移动意味着需对当前场景进行重绘，相对于场景的贴图位置，该过程需要计算相机的新位置。
- ☑ 可通过硬件缩放机制（如果存在的话）或对贴图渲染加以调整以实现缩放效果的复制功能。另外，在传统的卡通动画中，各层可能会移出显示范围之外。缩放效果还可通过移动分层背景进行模拟，该方法常出现于传统的卡通动画中。相关示例可参考[Thomas95]。
- ☑ 2D 贴图的旋转操作较为困难，这需要强大的图形处理性能作为后盾。某些早期游戏机游戏在其硬件设备上提供了相应的支持，进而模拟旋转操作以及透视比例问题以替代原有的贴图方

案（例如 SENS，即人们所熟知的 Mode7）。而其他游戏则在位显示缓冲区渲染数据，并于随后通过硬件自动转换为贴图（如 SEGA CD 游戏机游戏）。这里，存在一类硬件可支持软件内的独立像素的设置操作（如基于 SNES 的 Super FX 协处理器），且可通过硬件自动将数据转换为贴图模式。

实际上，2D 风格渲染需实现一定程度上的相机管理，尽管这一过程并不明显。例如，相机位置确定了可见的部分 2D 场景以及显示设备上的玩家角色的相对位置。如果需要对当前场景进行旋转操作，则相机的旋转可有效地对该属性加以控制，并确保较为重要的游戏元素得以正常显示。

技术要点

需要注意的是，即使某些表现风格自身定义为二维场景，仍需要采用三维坐标系统作为其内部表现方式。例如，变焦操作常用于模拟贴图上方相机的垂直位移。正等轴测投影（即平行投影）常视为二维状态，然而，游戏对象以及相机常使用其坐标中的垂直分量调整其与 2D 贴图之间的位置，并添加相应的深度感。

透视表现方式（3D）

根据人眼的感知系统，大多数游戏均采用一类固定的表现方式用以模拟三维场景世界，该方式称作透视投影。其中，平行线将汇于某一消失点。需要说明的是，根据人眼的观察位置，当场景内的顶点回投至平面上时，透视投影将导致最终图像产生变形。关于三维图像的生成方式以及各类投影的视觉效果，读者可参考[Akenine-Möller02]、[Foley90]以及[Pharr04]以获取进一步的信息。图 2.8 显示了基于透视投影的相关示例。

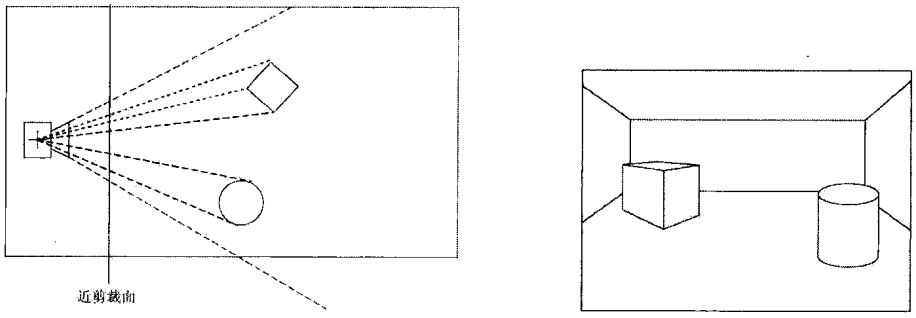


图 2.8　游戏场景中典型的三维透视投影

可调整游戏相机的渲染属性以增强或改变玩家对游戏场景的感知程度。例如，FOV 可极大地影响玩家相对于游戏场景的运动状态，修改相关数据值类似于调整真实相机的镜头。另外，FOV 还可用于控制视锥体的尺寸以及游戏场景的渲染范围。增加 FOV 后，渲染视野也随之变得更为开阔，如图 2.9 所示。该技术常用于强调以偶像场景中快速移动的物体，在一段时间内，可对 FOV 进行适当的调整，进而模拟物体的加速状态。相反，减少 FOV 则可实现游戏元素的近距离放大效果，该效果可模拟狙击枪的瞄准镜。

另外，3D 表现风格还包含下列显著特征：

- ☑ 可通过透视投影、正交投影、轴测投影以及其他投影方式渲染 3D 场景，具体内容将在第 4 章中加以讨论。
- ☑ 游戏场景的投影结果可根据游戏状态动态改变。

- ☑ 与 2D 相机相比，3D 相机具有更加强大的真实感并可大大提升玩家的游戏体验。
- ☑ 3D 相机根据人眼感知世界的方式对游戏场景加以描述。
- ☑ 3D 相机很少用于生成立体视觉场景（出于性能原因），因而无须佩戴特制的立体视镜。

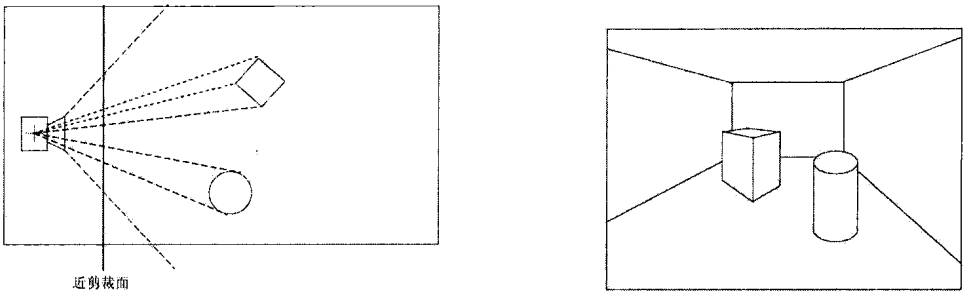


图 2.9　FOV 可视为一类重要的渲染属性并可对其实施动态调整

混合表现方式（2.5D）

相对于二维处理方案，2.5D 显示风格的最大差异在于：尽管游戏的主体部分采用正交投影加以显示，但仍然存在某些元素用以模拟三维场景，其中包括部分背景内容以及游戏对象。通常情况下，相机的运动方式平行于正交投影线，进而突出一种准三维效果。需要注意的是，2D 与 3D 元素间的混合方式包含多种变化，某些游戏采用 3D 背景并辅以 2D 对象，例如 Paper Mario（[Paper00]）。

相机的表现风格还可通过视差效果进一步增强三维场景世界的视觉效果。此时，应对相机的位置加以严格控制，并被限制在预定位置或位置之上。

正等轴测投影则是一类较为常见的平行投影，轴测投影的一种形式即是使用世界坐标轴且对应轴与水平方向呈 30° 角（如图 2.10 所示），该投影将生成一类准 3D 场景，通常情况下，可采用预渲染背景贴图并辅以预渲染或动态渲染精灵以匹配当前投影方案。鉴于这一类准深度感，正等轴测投影常称作 2.5D 投影。

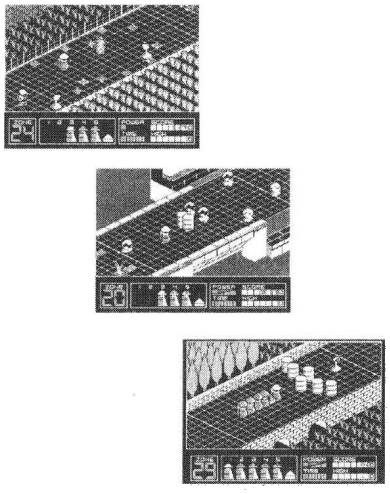


图 2.10　基于 2.5D 的正等轴测投影示例。游戏 Highway Encounter 由 Vortex Software 于 1985 年发布

针对于缺少 3D 硬件加速的目标平台而言，正等轴测投影可大大改善渲染效果，尽管玩家的场景视野调整能力依然受到限制。通常，在内存条件允许的情况下（可针对各潜在的视野方向存储背景图案以及精灵对象），虽然这一方案可显示 4 个不同方向上的视见内容，但该选择项并不在玩家手中。多个正等轴测投影视见内容则需要额外的设计工作以确保视见内容间的一致性（贴图独立进行存储）以及正确性。

相机行为

如不考虑相机的表现风格，游戏相机的行为可划分为 3 种类型，即视频相机、第一人称相机以及第三人称相机，下面将针对各种相机类型加以简要分析。除此之外，还将讨论其他两种相机行为，即反馈相机和预视相机。

视频相机

在本书中，游戏场景中的视频相机定义为一类非交互式相机，该类型相机的背后技术实现方式与其他相机类型基本相同，其主要差别在于：所显示的游戏场景画面内容往往不在玩家的控制范围内。需要注意的是，玩家仍然可对游戏内的视频事件以及实时视频事件进行操控，当然，这也会涉及其他具体问题（例如，期望的视频事件应对玩家真实可见）。第一人称视频相机十分常见（常出现于影视业中并用于生成各类视觉效果）。视频相机序列有时也称作视频动作，该术语表明，视频序列实时生成且无须考虑观察者与当前游戏的交互状态。下面将考察两类常见的视频相机应用。

非交互式过场动画

针对玩家而言，过场动画类似于影视场景画面，该术语反映了早期游戏中的某些技术限制，亦即，无法提供较为简单的场景切换方案。具体而言，玩家的交互性以及相机行为的调整皆被锁定，连续的画面间不允许存在任何修改操作。

当然，类似于影片的放映过程，此处仍可向玩家提供相应的信息以及说明性内容。同时，还可采用相关视觉效果以进一步丰富画面内容。需要注意的是，某些视效可能会受限于性能、内存以及游戏的实时特征，因而无法得以全面展示。

在某些情况下，宽高比、视口尺寸以及其他渲染特征均会产生不同程度的变化，进而向玩家展现某种非交互式特征，相关内容还包括纹理滤波以及音频效果，以凸显其与正常游戏进程之间的差异。

过场动画功能用于提供相关故事情节或游戏信息，且应至少向玩家显示一次。同时，这一类动画应包含相关选项并允许玩家跳过当前视屏播放。

实时视频序列

类似于电影放映过程，实时或游戏内的视屏播放序列常蕴含着相关游戏事件，这一类视屏通常在正常的游戏环节中连续播放且不受玩家行为的干扰。与非交互式视频序列相比，其主要差别在于，若相机位置不当，则无法保证玩家可查看到当前发生的游戏事件。若相机的应用状态发生改变并开始播放非交互式视频，应确保玩家角色可见以及动态相机位置所导致的控制操作的正确性。

通常情况下，上述视频序列常用于增加画面的真实感，其中，相关事件持续进行且不受玩家操控的影响。亦即，玩家无法与视频序列中的相关对象发生互动，进而避免未知变化对脚本事件产生不良影响。

第一人称/视点

传统的第一人称（FP）相机模拟人眼的观察方式向玩家展现游戏场景画面，因此，这一类相机也称作视点（POV）相机，图 2.10 显示了典型的 FP 相机示例。某些较为著名的游戏均采用了第一人称相机，其中包括 DOOM [DOOM93]、Quake [Quake96]和 Halo[Halo01]。

针对游戏场景中的玩家角色人物，第一人称相机提供了强大的临场感以及真实感。实际上，该类游戏的划分方式亦与相机类型紧密相关，即第一人称射击游戏，简称 FPS。甚至，某些采用 POV 相机的不同风格的游戏也可划分为同一类型。

玩家角色通常隐匿于 FP 相机中，这在某种程度上可降低游戏的临场感，当然，可添加相关视觉元素对此加以改善。较为常见的处理方案是显示武器装备或玩家的部分肢体，例如胳臂。需要说明的是，对应的视觉显示位置无须过分精准，相关示例如图 2.11 所示。其他视觉元素还包括玩家头盔、HUD 等道具。另一种常见方法是适当地对渲染画面加以偏移，进而模拟移动时的视觉震动，例如相机的震动可用于对相关游戏事件加以反馈。类似地，还可使用音频元素以强调音效、回声以及脚步等内容的声音定位。

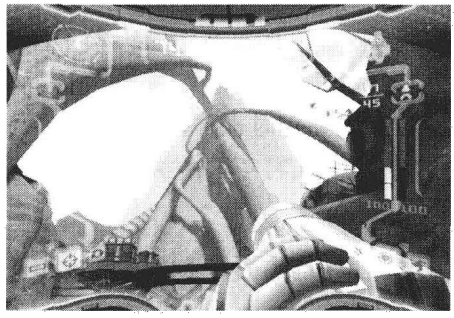


图 2.11　第一人称相机视角。图像源于 Nintendo 公司于 2004 年发布的游戏 Metroid Prime 2

另外，FP 相机还可置于飞机的驾驶员座舱内并添加相关视觉元素（包括座舱环境模型以及电子显示设备）以进一步强调玩家在游戏场景中的位置。该方案常出现于赛车游戏以及飞行模拟器中，此时，玩家往往不会察觉到 FP 相机的存在。

然而，FP 相机依然存在某些潜在问题。与第三人称相机相比，第一人称相机的视野往往会受到很大的限制。例如，FOV 相机常会体现出某种局限性且玩家难以在其位置间作出正确的空间关系判断，这也意味着，若相机或玩家的位置未曾采取适当的调整（直接由玩家加以调整或游戏系统自动调整），则很难观察到其近处的场景对象。针对于此，第一人称相机往往很难实现精准的跳跃动作。

当玩家移动或执行其他游戏动作时，FP 相机的方向性操控通常比较复杂，这一般取决于控制器设备，即某些物理硬件布局支持移动和方向上的组合操控。此处，较为常见的例子是基于第一人称 PC 游戏中的鼠标和键盘组合。

在车辆模拟器中，FP 相机常与各种第三人称相机视点结合使用。其中，主视口可视为第三人称相机，尽管该视口代表了基于驾驶位置的玩家视点。这里，主要差别在于，相机可执行围绕前向轴的旋

转操作，在游戏环境中，如无特殊要求（如玩家角色人物倚靠于墙角附近），FP 相机通常会禁止该操作。

技术要点

在 FP 相机的实现过程中，尽管相机的位置与玩家角色头部位置相关，但在实际操作过程中，二者应彼此独立，其原因在于，当玩家在游戏场景中跨越较小的障碍物时，垂直轴向运动方向上的延迟将对视觉效果产生不良的影响。否则，相机（玩家）将对每次细微碰撞均做出反应，从而导致相机的不连续运动。除此之外，还应确保相机的旋转操作脱离于玩家角色的方向，并提供自动倾角控制或来自玩家的全视角控制。

第一人称相机还可采用全屏（或全视口）过滤效果以调整场景画面，例如模糊效果以及用于模拟水下区域的失真效果。除此之外，相关操作还包括屏幕相关空间内的对象渲染，例如头盔外表面的水渍，头盔玻璃表面反射出的玩家脸部特征。上述效果可在相关画面内容渲染完毕后执行，即所谓的后处理操作步骤（位于头盔或 HUD 渲染操作之前），抑或与渲染操作结合进行，进而模拟基于对象或视觉画面的各种相关属性（例如红外线热度信号）。

第三人称/分离视角

大多数 3D 游戏相机的置点通常与玩家角色分离，一般位于其后方或上方，这一类相机称作第三人称（TP）相机。此时，场景画面的渲染位置将位于游戏角色人物的外部。在 3D 渲染环境中，Super Mario 64 [Mario96]是一款较早采用该表现方案的游戏之一，并可视为该类游戏的经典实现。早期游戏常包含明显的街机风格，例如“I，Robot” [IRobot83]。然而，这一类游戏均可视为第三人称游戏，尽管其表现风格定位于 2D 游戏。

与 FP 相机相比，第三人称相机的临场感则稍显逊色（部分原因在于：玩家角色模型可见），但这丝毫无法掩盖其诸多优点。与玩家角色人物相比，尽管相机的相对位置以及方向较为灵活，但 TP 相机可清晰地反映出玩家角色与环境之间的关系且兼具交互性。因此，FP 相机可视为一类较为常见的游戏选择方案，并可满足场景漫游这一类较为重要的游戏体验元素以及前面所讨论的平台（或滚屏）类型。

基于上述方案，玩家可针对灵活多变的位置数据以及方向数据实施动态选择，根据游戏体验需求，玩家还可自行选择场景视角。相机视野内容可实现即时改变抑或根据设计者的期望要求实施插值计算。除此之外，相对于玩家角色人物的运动方向，相机还可动态地调整其距离以及角位，而非在预定义组合数据间实施简单的切换插值，具体内容如图 2.12 所示。该方案常用于玩家角色人物占据主导地位的游戏环境中，如动作冒险类游戏以及角色扮演类游戏。

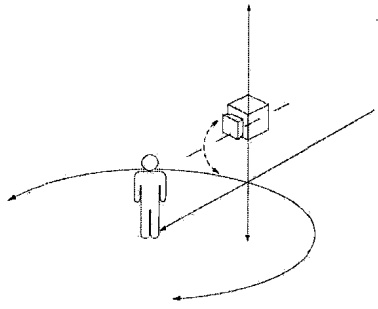


图 2.12　玩家可对相机距离、高度位置以及方向实施动态调整

然而，在某些游戏环境下，相机会移出当前场景范围，抑或特意展现某些应被隐藏的游戏元素，

因而需禁用上述控制操作。偶尔，玩家还会察看场景中的某些特定游戏对象，因而对相机实施限制还是十分必要的。

除了基于玩家角色人物的位置和方向定义相机之外，还存在其他方案用以确定第三人称相机的位置，如静态相机或某些受限于固定路径上的相机。在某些情况下，建议采用一致的相机解决方案，并采用有别于常规相机表现方式的某种特定处理方案体现相关游戏场景（如游戏中的关键人物 Boss 或其他特定场景）。考虑到敌方角色的物理特征以及需要充分展示的某些游戏对象，该方案是十分必要的。另外，根据特定的环境以及基于相机对象的潜在碰撞问题，对相机进行某种限定依然不可或缺。

如前所述，实现 TP 相机并非易事，其中涉及两个主要问题，即场景漫游以及碰撞检测。亦即，相机应能够在复杂（或动态变化）的场景环境中自由地穿越，并有效地解决与游戏对象或场景几何体之间的碰撞问题，如图 2.13 所示。除此之外，相机还可能被置于某一特殊位置并以一种特定方式显示当前场景。在很大程度上，这将取决于场景环境的物理布局，因而需要对相机的可能位置加以限制。该问题常与玩家人物角色的行为特征相关（如加速运动或跳跃运动），当然，可通过预定义的相机位置或移动方式减少或避免这一类问题。然而，游戏需求中应包含相应的动态解决方案。需要说明的是，上述问题通常较为复杂，在某些情况下，还需要针对具体的环境结构做出适当的调整。

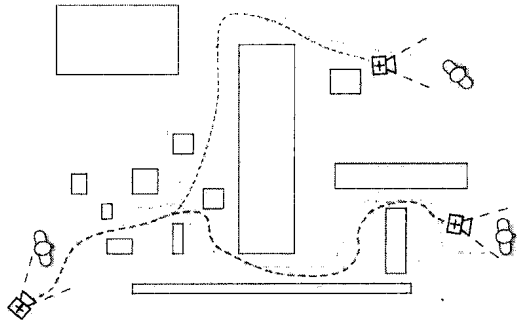


图 2.13　复杂场景中的漫游操作，该操作实现起来并非易事

另一个常见的问题是玩家角色人物的遮挡现象。大多数游戏允许玩家移至被当前环境所遮挡的区域。其中，较为常见的是部分遮挡以及临时性遮挡。然而，全部遮挡常会导致玩家失去方向感，这在紧张的战斗场景中将会是一个十分严重的问题。

另外，TP 相机内的视线问题也颇具意义。在大多数情况下，玩家并非总是沿其与目标对象之间的视线方向进行观察，某些视线元素（如场景中的 2D 或 3D 标线）、火力跟踪装置等内容都将会大大地方便玩家的定位操作。同时，自动瞄准装置（无须玩家输入控制）、辅助瞄准装置（玩家需要输入操控的支持，但其精确度无须达到像素级）以及自动定位装置均可大大提高玩家的精准度。第 4 章将对玩家与相机操控之间的关系加以进一步的阐述。

无论如何，TP 相机值得关注。TP 相机不仅可在游戏场景（包含丰富的图形元素，如动画）中充分展示玩家（或该玩家所控制的角色）的存在，还可清晰地显示玩家在场景空间中的位置状态以及相机的定位和移动方式，这一点是真实相机设备无法比拟的。除此之外，如果加以正确的利用，第三人称相机还可向玩家提供信息线索。然而，若相机无法得以正确的实现，或无法完美体现交互式游戏中所蕴含的动态变化特征，则玩家的游戏体验将会大打折扣。

针对于全部潜在动作，TP 相机的选择方案应支持玩家角色人物所包含的动画操作。考虑到相机的

表现风格，这将会产生另外一个问题，即如何确定相机的方向。角色或动作类冒险游戏通常会将目光集中于游戏角色本身，因而相机的位置往往设置于角色的上方。无论人物角色采取何种动作，相对于显示屏幕，都将保持同一位置。当角色人物呈现为静止状态时，该方案工作良好；一旦角色人物开始移动时（尤其是以较高的速度运动时），该方案对于运动方向上的、基于玩家角色的前视问题将十分有效。同时，这也使得玩家可以更好地对未知环境加以预测，并拥有充分的时间对相关游戏事件（如敌方火力进攻）加以反馈。图 2.2 显示了基于前视问题的相关示例，具体内容还将在第 7 章中加以讲解。

针对于游戏（或视频播放序列）中的大多数相机风格，第三人称相机包含了丰富的内容以及多变的位置以及方向行为，在下面的小节中，读者将会看到大量的相关示例。

后定位

相机的位置可根据角色人物的方向加以确定（通常情况下，可置于角色人物的后上方以避免该角色削弱当前游戏场景画面），这将导致沿角色垂直轴向上的旋转延迟（如图 2.14 所示）。

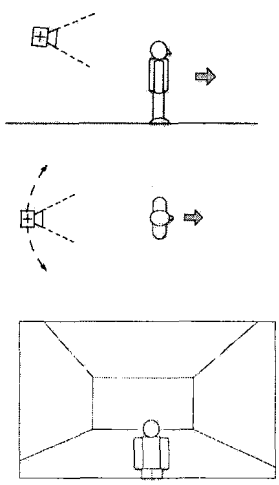


图 2.14　基于玩家角色人物的后定位

肩部定位

相机可定位于角色人物的肩部并据此生成对应的视见内容，这将降低或完全消除延迟现象（如图 2.19 所示）。该方案常见于第一人称/第三人称混合相机模式中，以提高玩家的瞄准精度。

脚部可见

相机的定位远离当前角色人物并保持动画清晰可见，且周边区域依然可获得良好的视野。同时，该方案还可令角色人物的脚部可见，这对于判断跳跃动作或其他角色定位方式是十分重要的（如图 2.15 所示）。该类相机的行为方式类似于前述后定位相机模式，如图 2.14 所示。

远距离定位

相机位于玩家角色人物的远后方，该方案与当前场景环境紧密相关且常见于赛车类游戏中。此时，相机的方向常落后于角色人物的移动状态，进而可消除快速重定位所产生的负面影响，并使得当前角色人物呈现为三维状态，如图 2.15 所示。

角色相关

这里，相机将根据角色人物的位置加以定位（常用于回放型相机）。例如，相机可定位至对象的尾部或侧面（飞行模拟器）、防护板（赛车）等。相关内容如图 2.16 所示。

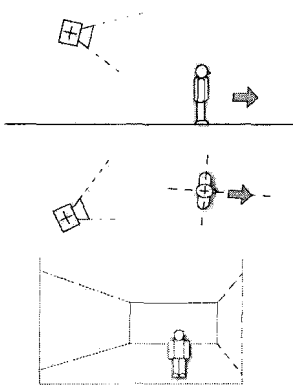


图 2.15　采用中距离时，玩家角色人物的脚部可见

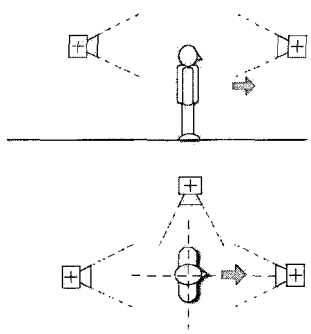


图 2.16　角色相关

静态相机

相机通常定位于某一（较高的）有利位置并呈现为静止状态，必要时，还可实现重定位以对玩家角色人物或其他目标对象进行跟踪（如图 2.17 所示）。

路径定位

此处，可对相机加以限制并使其沿场景中的预定义路径行进，相机沿路径方向上的移动位置则由玩家角色人物与参考帧之间的相对位置加以确定，也可能是相机路径自身（如图 2.18 所示）。

与 FP 相机相比，TP 相机使得玩家对其自身的位置和方向有着更为明确的认识，然而，在较为复杂的场景环境中，确定正确的相机位置和方向进而获取清晰的视见内容则并非易事。

这里，对于 TP 相机的考察重点不仅包含相机与玩家角色人物之间的相对位置，相机的方向性问题也不容忽视。多数第三人称相机的表现方式则是将相机与玩家人物直接加以绑定，开始时，该方案可获得较好的场景画面，但稍后读者将会看到，基于玩家人物的前视相机也不失为一种较好的解决方案。

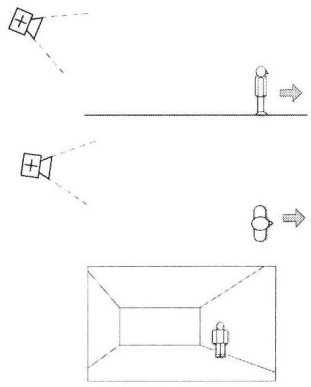


图 2.17　静态相机，用于跟踪玩家角色人物

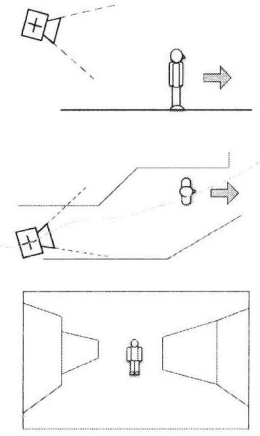


图 2.18　路径限定

技术要点

相机系统中 FP 相机以及 TP 相机的内部表现形式以及渲染系统中的游戏画面表现方式并无太大差异，同时，这两类相机一般继承于某一公共基类。若 FOV（或者远、近剪裁面）基本一致，则用于将坐标从相机空间转换至屏幕空间的数学转换矩阵（如投影矩阵）也基本相同。在多数情况下，表现方式之间的差异主要源于基于玩家角色人物的相机位置以及玩家操控的状态变化。

混合 FP/TP 相机

某些游戏从外部位置对角色人物进行观察，以在保留第一人称相机操控、定位能力的同时改善 FP 相机的输出画质。通常情况下，该方案可将相机定位至角色的肩部位置，尽管实际位置将根据游戏的动态需求有所出入。采用这一类相机方案的游戏包括 Jet Force Gemini[JetForce99]、Resident Evil 4[Resident05]以及 Gears of War[Gears06]。图 2.19 显示了常见的可用位置示例。

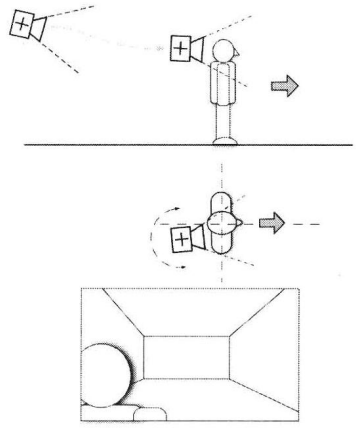


图 2.19　混合 FP/TP 相机方案将根据玩家角色需求调整相应的位置

另外，相机的位置甚至还会由具体的游戏状态加以确定。在某些游戏中，相机一般采用常规的第三人称位置，直至玩家装备武器或进入“动画模式”位置。此时相机将近距离接触玩家角色人物。另外，视点也会发生变化以实现玩家角色与游戏场景元素之间的交互（如游戏 Metroid Prime 3 [Metroid07] 中的控制杆等）。针对于 TP 相机，类似的位置变化还将发生于玩家返回至碰撞几何体时，进而对游戏体验实现良好的支持，抑或消除几何体间彼此穿越这一现象。

技术要点

当玩家角色与相机之间近距离接触时，应确保玩家角色模型与相机视锥体近剪裁面之间处于分离状态，以消除渲染过程中所产生的负面效果。另外，Z 缓冲区的分辨率很可能无法实现玩家人物角色的准确排序，因此需要采用其他渲染方案（例如，忽略深度值排序或使用特定的排序方案）。另外，即使相机完全位于玩家角色碰撞体内部，考虑到视锥体的形状，近剪裁面与场景之间若处于相交状态，则渲染过程依然会出现问题。当玩家接近渲染几何体时，屏幕边缘可能会产生上述问题。如果有必要移动相机以防止其与几何体之间的碰撞问题，应适当地调整相机的方向以使其指向基于玩家角色的前向位置。

针对于 TP 相机中常见的交互性问题，混合相机可视为一类有效的解决方案，并且还可进一步提升玩家的游戏体验，尽管在一定程度上视野内容仍然会位于玩家角色外部。但与 TP 相机相比，相机漫游问题得到了很大程度上的缓解，当相机位置位于玩家碰撞体内部时尤是如此。同时，基于玩家角色的相机旋转延迟问题也得到了很好的解决。

当游戏中采用基于全视角定位的大型武器模型或类似控制方案时，FP 相机操控以及 TP 相机定位所形成的混合相机模式可视为一类有效的解决方案。

技术要点

针对于具体的实现方案，第三人称相机体现了各种相机类型的操作行为，实际上，前述第一人称相机可视为第三人称相机的一个子集，并辅以与位置、方向相关的特定限制条件。如果不考虑相机间的表现形式，视见变换矩阵、可视计算以及相关相机信息的访问过程基本上大同小异。针对相机系统，第一人称相机和第三人称相机可通过继承某一公共相机基类得以实现，这也是较为常见的实现方式。关于相机的实现方式以及对应的替换方案，具体内容可参考第 11 章。

预视相机和反馈相机

大多数游戏相机均可描述为反馈型相机，亦即，相机将根据玩家人物的位置、方向以及其他属性反映其当前（或之前）的状态。而预视相机则用于分析玩家角色和其他相关游戏对象的可能行为，并对最佳位置和方向加以预测，进而提供较好的玩家事件内容和环境画面。除此之外，上述两种相机类型均可消除场景几何体遮挡、对象碰撞以及其他未知事件所导致的潜在问题。这通常可有效地预测玩家的位置、游戏对象或生物相对于当前场景世界的状态信息，甚至还包括相机自身的信息。

考虑到与玩家角色之间的依赖性，大多数 FP 相机需实现为反馈型相机，并需要借助于相应的预视技术。例如，当玩家呈现为倾斜状态时，相机应自动调整其高度状态，具体内容将在第 7 章中加以探讨。而第三人称相机多实现为纯反馈型相机，因而预视技术的重要性不言而喻。

是否仅预视相机可有效地解决诸如复杂场景漫游这一类问题目前尚存争论。然而，考虑到玩家输入状态固有的随机特征，该方案很大程度上取决于动作类型、目标角色方向和/或速度的变化率以及玩

家所处的环境。玩家行为分析将有助于上述问题的解决，但更为重要的是，读者应了解物理特性对玩家角色移动所产生的种种限制。下面将逐一讨论相关的预测技术。

预测人物的移动方式

尽管与游戏类型紧密相关，玩家角色人物位置预测在相机逻辑中仍占有重要的地位。该操作主要出现于 TP 相机中，其中，相机将在复杂的场景环境中实现漫游。除此之外，该预测技术对于相机最佳方位的确定也具有十分重要的意义。

当处理器的计算能力足够强大时，游戏模拟可对其未来状态实施准确的预测，该策略更适用于网络游戏（其延迟问题常需要针对物体的未来位置加以预测）。实际上，相应的简化计算（如基于目标对象的速度）即可对目标对象的期望位置实施较好的近似预测。然而，该方案在很大程度上取决于当前目标对象的移动特征，特别是方向变化及其加速状态。大多数人物角色的方向以及加速变化通常较为缓慢，这将有助于实施上述计算。亦即，当玩家角色的移动方向迅速变化时，通常会增加预测计算的难度。

针对预测计算，车辆以及其他高速运行的物体则更容易出现问题。然而，针对于未来位置的变化，可在相机反馈过程中添加相应的延迟，以使相机的移动和方向的调整更好地适应玩家角色动作。

[Halper01]对移动预测的优点进行了详细的论述。在图 2.20 中，玩家角色在两点之间采取直线运动，通过对相机位置进行预测，相机系统可顺利地行进至下一位置并可有效地避免潜在的碰撞问题。在较为复杂的场景环境中，当需要对未来状态信息实施准确预测时，该过程还会引发其他问题并产生较高的性能开销。

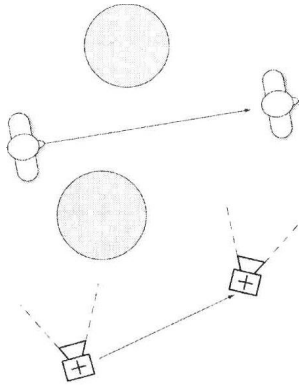


图 2.20　与反馈型相机相比，预测型相机可方便地实现场景的漫游

预测相机的路径

相机的移动常被限制在某一预定路径之上，该路径通常由场景编辑工具加以定义。然而，在某些情况下，移动路径也可动态生成以对当前游戏状态做出正确的反应，例如几何体间的避让行为。动态路径的相机移动问题常出现于 AI 的实现过程中，并具有显著的特点。实际上，TP 相机即可视为一类 AI 代理，并根据其移动方式以及方向状态满足某些外观要求。

路径搜索方案

当动态确定相机路径时，可将该问题分解为下列几项内容：

- ☑ 确定全局路径。搜索当前场景并确定执行目标位置的适宜路径，同时还应保证观感符合一定的要求，如与场景元素之间的距离等。
- ☑ 确定局部路径。例如游戏对象、岩石或柱状躲避物。
- ☑ 基于目标对象的搜索。亦即，在各种限制条件下获取基于目标角色的最佳位置，如阻挡物体、几何体躲避物体等。

数据驱动与规划方案

这里，针对于数据驱动或预处理方案的内存开销以及实时相机位置的动态处理，应选取相应的折中方案。除此之外，当采用实时处理或某种规划方案时，将对 CPU 的性能提出较高的要求。

方向预测

如前所述，通过对目标对象的位置进行预测，相关结果可用于确定相机的未来方向。然而，若过早对“理想”方向执行插值计算，抑或预测计算过于超前，则结果方向往往会对当前游戏状态产生负面影响。

这里，需要注意的是角度插值计算。当在两个既定方向之间执行插值计算时，可简单地使用线性插值或球面线性插值得以实现，进而获得最终的方向定位。此处，位于当前方向与期望方向之间的角度随着时间的推移将不断减小。如果目标方向和原始方向在一段时间内皆发生变化，则需要采用其他辅助步骤以计算插值结果，其原因在于，原始角度和结果角度将会随着时间的推移不断增加且与插值计算无关，当角度重定位速度较慢时尤是如此。如果新角度正比于初始角度和结果角度之间的角度值，则与前一帧相比，当前相机方向和结果方向之间的角度值将会增加。另外，如果当前角度正比于初始角度和结果角度之间的原始角度值，则最终结果将根据结果角度的变化速度呈现为不连续运动状态。

为了确保插值计算随着时间的变化呈现为平滑状态，此处应从目标位置至初始位置实施逆向计算，通过该方式，两方向间的夹角将随着插值过程的不断进行而逐步减小。

视见生成

当读者了解了与游戏相机相关的基础理论后，如何通过相机产生当前游戏场景的视见内容？诸多计算机图形学专著均对此进行了全面的阐述（即渲染管线），例如[Akenine-Möller02]和[Foley90]，如果读者已具备这方面的知识，则可跳过本小节的内容。了解视见内容的具体生成过程对设计者而言大有裨益，当然，这并非是硬性要求。

根据当前图形硬件的特征，视见内容的生成过程可分为多个不同的阶段，某些阶段可通过常规的

游戏逻辑或在图形硬件中得以并行执行，如下所示：

（1）游戏逻辑负责更新处于活动状态下的全部游戏对象的内部状态，包括位置的确定、方向状态、动画状态以及其他对象的状态变化。

（2）游戏相机包含自身更新逻辑，以确保与当前视见内容相关的位置、方向以及其他属性处于正确状态。

（3）针对所生成的各种游戏视见内容（如图 2.21 所示）。

① 确定游戏相机的位置和方向。

② 确定显示设备的视见部分（即视口）。

③ 根据当前相机的位置和方向，确定视见体或视锥体进而显示当前游戏场景的可见部分。

④ 待视锥体定义完毕后，处于渲染目的，可对游戏对象或其他元素执行剔除操作。

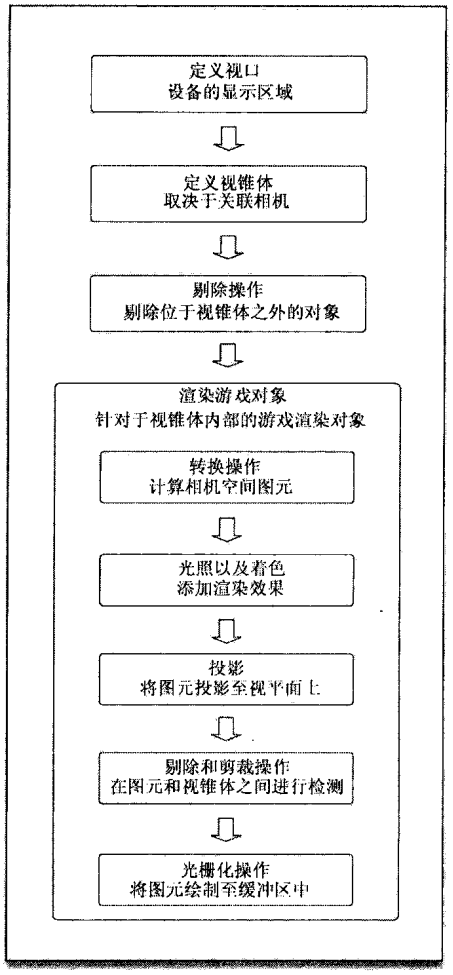


图 2.21　简化后的图形渲染管线

⑤ 针对各种游戏对象或场景元素。

☑ 根据当前相机视见部分对游戏模型（亦即，用于生成其外观形状的几何元素，如三角形或四

边形）实施转换操作。

☑ 应用光照和着色效果。

☑ 将三维图元投影至二维平面之上。

☑ 针对位于视见体外部的投影图元实施剔除操作以及相交剪裁操作。

☑ 对图元执行绘制操作（亦即光栅化操作）。

需要注意的是，上述内容仅包含了某些较为典型的操作步骤，某些内容可能会根据特定的渲染系统发生变化（如图像硬件的计算能力、光线跟踪技术、光栅化操作等）。

视锥体

视锥体可视为一种金字塔结构，其顶端被平行于底面的平面加以切除，如图 2.22 所示。当然，其他形状的视锥体也并无不可，它们常用于模拟不同种类的相机镜头。此处，若将视锥体视为一种金字塔结构，则相机定位于金字塔的顶端，视锥体的中心线则表示为相机的视见方向。其中，视锥体的矩形平面亦将随相机的转动围绕当前轴加以旋转。

在视锥体中的两个矩形平面中，距离相机位置最近的平面称作近剪裁面，而远端平面则称作远剪裁面。这里，位于远、近剪裁面之外的对象均不予渲染。

需要说明的是，游戏相机的 FOV 对应于真实相机镜头设备的可见视野。在相机系统中，视域可定义为垂直或水平方向上的角度值，这一类角度进一步确定了当前视锥体的形状，如图 2.23 所示。当已知某一角度时，可根据视野的宽高比值计算另一个角度。

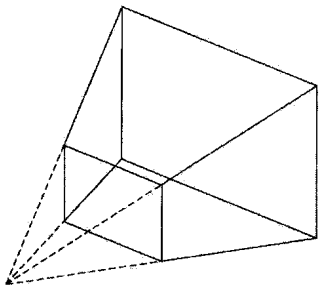


图 2.22　视锥体呈现为金字塔形状

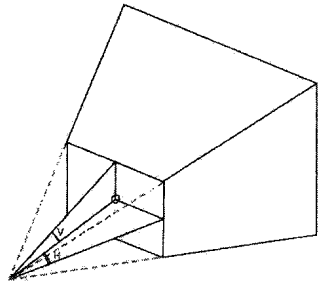


图 2.23　水平（H×2）和垂直（V×2）视域角定义了视锥体的形状

视见转换

视见转换可视为一类数学构建过程（通常情况下会使用 4×4 矩阵），作为渲染过程中的某一操作步骤，该过程将世界空间坐标转换至相机空间内。在游戏相机系统中，针对于当前场景相机，视见转换将根据其已知位置和方向加以构建。

技术要点

考虑到视域将直接影响游戏场景的可见部分，因而常用于生成长焦镜头效果并可通过降低视域范围得以实现。相反，增加视域将扩大当前场景的可视范围（当然，这也会随之产生额外的计算开销），但随着视域角不断增大，该过程也会

产生某种副作用（该效果在进行直线混合操作时尤为明显）。除此之外，远、近剪裁面之间的距离也会极大地影响深度缓冲区的精确度，其中，深度缓冲区常用于场景图元正确的绘制顺序。随着远、近剪裁面间距的不断增加，远景对象可能会出现错误的排序状态，抑或与期望中的排序结果大相径庭。

技术要点

通常情况下，相机的位置和方向往往预置于矩阵格式中，因而可直接运用于相关计算过程中。鉴于相机矩阵将本地相机空间转换至世界场景空间内，因而视见转换一般为相机矩阵的逆矩阵。然而，常见的情况是，视见转换常根据需要使用基于世界空间坐标系的 LookAt()函数加以构建（可不将相机数据按照矩阵格式加以存储）。这里，LookAt()函数可视为相机数学运算的基础，该函数将根据原始位置、目标位置以及场景的 Up 方向向量构建转换矩阵。需要注意的是，该计算过程仅将游戏相机重定位至目标方向，并不生成最终的渲染转换。在渲染的前期操作中，对象将被置入相机空间内，并于随后传递至渲染机制中。第 10 章将具体探讨游戏相机的重定位功能，其中将使用到 LookAt()函数。

投影转换

投影转换可视为另一类数学构建过程（统一采用 4×4 矩阵），其中，相机空间坐标将被映射至单位立方体中。因此，该步骤执行完毕后，坐标范围将位于−1～+1 范围内，即标准化设备坐标，与单位立方体外部边界相交的图元将被剪裁并生成新顶点。随后，该坐标将被转换为屏幕空间坐标并传递至光栅化处理阶段。这里，存在两类投影方案，即平行投影和透视投影。

技术要点

透视投影矩阵的构建过程通常包含以下数据内容：

- ☑ 视域（抑或水平或垂直方向上的夹角）。
- ☑ 宽高比。
- ☑ 近剪裁面距离。
- ☑ 远剪裁面距离。

平行投影所需数据内容则稍显不同，具体如下：

- ☑ 近剪裁面距离。
- ☑ 远剪裁面距离。
- ☑ 视见宽度。
- ☑ 视见高度。

针对于平行投影，由于投影方向平行于相交本地空间内的前视方向，因而仅需要宽度值和高度值用以确定视锥体的各面，其投影矩阵的具体操作过程，读者可参考[Foley90]。

玩家控制操作

作为游戏设计的重要内容之一，由于玩家角色的控制操作依赖于游戏的表现风格以及行为选择方案，因而常在初始状态下即与相机系统发生关联，玩家控制操作的详细内容则超出了本书的讨论范围，下面将对其进行简要的阐述。

针对于控制器输入状态的改变以及它对物体移动的影响方式，控制参考帧体现了二者之间的关系。

在典型的第一人称游戏中，控制参考帧通常直接对应于玩家角色的朝向（即角色相关），并常被限制于围绕世界坐标系 Up 轴的 2D 方向之上。在第三人称游戏中，控制参考帧常取决于相机与玩家角色之间的相对位置（即相机相关，如图 2.24 所示），在某些情况下则是相机的视见变换（屏幕相关）。在少数情况下，角色相关方案也可同于第三人称游戏中，尽管当角色朝向相机时会产生某些问题。世界相关方案则较少使用到控制参考帧，其中，控制操作取决于世界坐标且与玩家角色或相机位置无关。

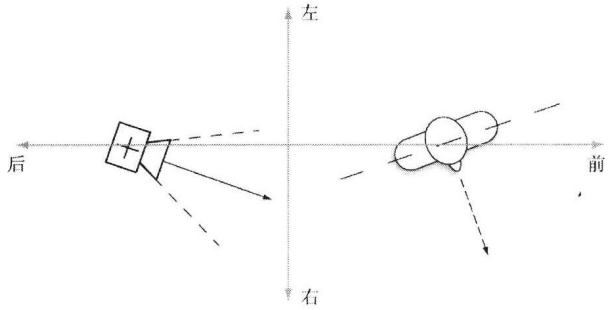


图 2.24　相机相关控制参考帧

在第三人称游戏中，控制参考帧常会出现无法预料的状态变化，当相机切换至某一新位置且该位置位于玩家角色的另一侧时尤是如此。针对于相机相关或角色相关控制方案，此时，玩家的控制器输入操作常会迫使相关角色返回至上一位置，其操控结果往往令玩家难以接受。

更为重要的是，控制参考帧应反映玩家的潜在动作，进而实现更为精准的定位。

本章小结

本章讨论了与游戏虚拟相机和其他实时应用程序相关的基础概念，及其与对应行为、属性相关的常见术语。不难发现，表现风格和行为的选择方案将极大地影响到玩家的游戏体验，第 3 章还将对此进行深入的探讨。根据玩家的喜好以及游戏体验需求，大多数游戏均组合使用了上述特征（即表现风格和行为特征）。针对特定的游戏环境以及当前技术条件，相机设计者应甄选出较为恰当的组合方案。

第3章 电影技术

虽然本书的重点在于实时相机的设计方案，但读者依然有必要了解相关的电影技术理论和规则。另外，游戏中的相机系统也会运用到电影视频方面的知识，这在常规的游戏产品抑或过场动画（即非交互式视频播放序列）中屡见不鲜。

电影技术的发展已日趋成熟并拥有一系列完善的规则，如场景合成、场景过渡、光照、相机走位等。在非交互式场合下，相关规则同样也适用于游戏中的电影视频播放序列。但是，电影技术如何应用于游戏环境中？游戏中实现电影技术是否适宜（一种理论认为：游戏视频相机应忠实体现真实世界中的对象，包括其物理特征以及相关限制条件）？早期的研究表明，游戏设计不应低于对应美学理论的下限。然而，动态游戏中相关技术方案的应用并非易事，在很大程度上，这将取决于游戏的风格以及具体的设计环境。

这里，存在若干选择方案以实现游戏中的视频技术；然而，向影视作品中添加相应的游戏元素则并非易事。例如，真实摄像机的诸多限制条件（如胶片的曝光特点）对于游戏设计人员而言并不存在，因而游戏相机可实现全方位的移动。随着主流电影中计算机图形技术（即 CG 技术）的不断发展，上述限制逐渐势微且终将被打破。针对于此，一个较为明显的例子是电影《黑客帝国》中的子弹飞行效果（The Matrix，Warner Brothers，1999）。在电影画面中，物体的速度减慢，而摄像机仍以正常速度前行，因而原本快速运动的物体其细节内容将体现得淋漓尽致。实际上，该技术在实时游戏中实现起来并不复杂。其中，可适当地调整游戏对象的运动速率并对相机实施全方位的操控。另外，针对于相机物理限制条件，其自由度的不断增加很可能会导致新兴电影技术的出现。目前，数码编辑技术、视频制作技术以及较为重要的显示技术已在影视作品中屡见不鲜。对于影视制作者而言，游戏技术自身亦可视为一类高效的视频开发工具。随着更为先进的三维视频显示技术的出现，影视以及实时游戏作品必将会大放异彩，届时，二者之间的差异将进一步缩小。

术语

电影技术涵盖了丰富的专业词汇用以描述摄像机的操作方式、场景的光照环境、演员的取景方案、摄像机的走位等。实时相机借鉴了某些电影术语并形成了一套专有的技术方案集。然而，在实时相机系统中，某些专业术语并未实现较为清晰的表达。例如，术语实时电影技术常用于描述各种不同的场景，其中包括交互式场景和非交互性场景——该表达方式稍显模糊。另外，当同一术语表达不同含义时，这也将给彼此的沟通带来很大的不便。下面将考察非交互式电影中的某些基础技术。

标准电影术语

较为系统的电影术语描述则超出了本书的讨论范围，读者可参考[Arijon91]和[Malkiewicz92]等专著以获取更为完整的理念。[Hawkins02]和[Hawkins05]则讨论了如何将影视技术应用于游戏设计中。除此之外，[Glassner04]还对某些相关主题进行了必要的补充，例如，故事大纲与游戏设计之间的整合方案。此处，某些主要的电影专业术语已然足够。

移动拍摄

移动拍摄是指摄像机在平行于水平轴 2D 平面间运动，若摄像机接近目标对象，则该方向上的运动则称之为前向推动；若摄像机远离目标对象，则该方式称作后向推动。相机的侧向（即相对于当前场景的 2D 方向）移动一般称作左侧航（crab left）或右侧航（crab right），如图 3.1 所示。

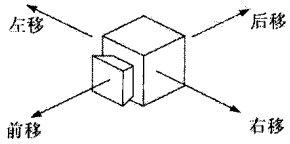


图 3.1　镜头的移动被限制在与地面平行的 2D 平面上

摇动镜头

该术语是指相机围绕世界坐标系统中的 Up 轴实现水平方向上的旋转，在大多数计算机专著中，这一过程则定义为偏航角。这里，二者之间具有相同的含义。需要说明的是，镜头的摇动并未定义任何运动，也未包含其他轴向上的旋转操作，如图 3.2 所示。

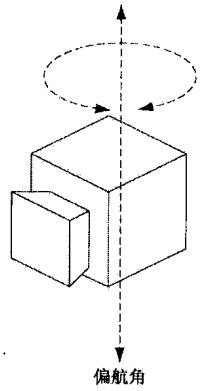


图 3.2　镜头的摇动是指相机围绕其 Up 轴旋转，有时也称作偏航角

镜头倾斜

镜头倾斜时，摄像机将沿其右向轴进行旋转操作，即摄像机垂直方向上的运动（该术语也称作倾角），如图 3.3 所示。

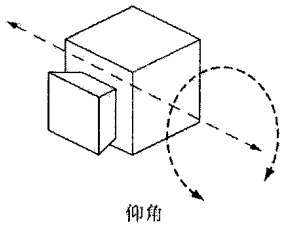


图 3.3 镜头倾斜时，摄像机将围绕其右向轴旋转，即仰角

跟踪拍摄

跟踪拍摄是指摄像机以一致的方向相对于另一物体进行运动。在游戏相机系统中，跟踪拍摄模式通常被玩家角色人物所代替，当然，最终情况还要依具体环境而定。当执行跟踪拍摄时，相机的移动常被限制在与水平面平行的某一平面上。在游戏中，大多数第三人称相机均可视为一类跟踪相机，如图 3.4 所示。

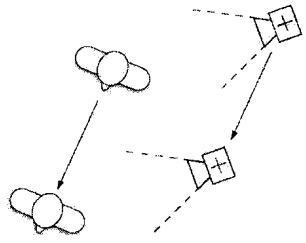


图 3.4 针对另一对象，跟踪拍摄通常会采取较为一致的相机移动行为

景深

根据相机的光圈的尺寸，真实相机设备通常会在某一设定距离上对物体实施聚焦。另外，该距离附近处（即近似焦距处）的相关对象也会处于聚焦状态（处于聚焦状态的物体将会呈现清晰的图像）。上述距离区域称为景深（DOF），且这一概念并不存在于实时游戏相机中。当然，可通过相应的渲染效果并对处于焦点之外的元素实施混合计算，即可实现景深效果。同时，DOF 常用于强调渲染场景中的特定元素，且无须对当前相机实施重定位操作：位于焦点处的相关对象自然会引起观察者的注意。相应地，改变角色间的聚焦状态则是一类较为常见的电影技术，该技术常用于转换观众的视线（突出人

物对事件或话语所产生的反应）。图 3.5 显示了基于 DOF 的相关示例。

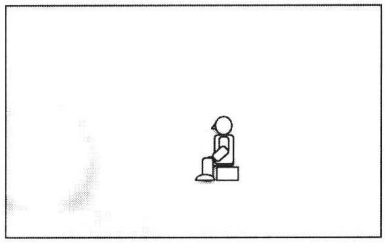


图 3.5　景深效果常用于强调某一特定场景元素，进而可动态改变观察者的视线

针对计算机渲染中的 DOF 模拟技术，读者可参考[Demers04]以获取更多的相关内容。

淡入/淡出效果

在色彩的淡入/淡出过程中，最终的颜色值将终结于某一固定颜色且通常为黑色（即淡出效果），具体情况也可能会有所出入。颜色的淡出效果常出现于场景结束抑或场景切换时，其本质可归结于线性色彩空间内的插值计算，或者亮度值以及 Alpha 值的混合计算。另外，当颜色值从黑色恢复至当前场景状态时，该过程则称作淡入效果。

镜头眩光效果

当相机镜头的摄入光线过于强烈时，画面通常会出现镜头眩光——通常情况下，这并非是一类期望中的视觉效果。一个较为典型的例子是，当摄像机摇向夕阳景色时，画面中常会生成眩光。镜头眩光常会使胶片中的影像变得模糊不清（或曝光过度）。另外，这一类效果还可能源于相机光圈的自身轮廓，此时，画面中常会呈现出一系列的彩色光环。大多数游戏均会力图模拟镜头眩光效果，进而使得场景的渲染结果更具真实性。该设计初衷值得称赞，但其具体实现过程往往会出现差错。例如，为了体现镜头的表现方式，通常会将相机置于游戏场景中，但这将使玩家的临场感大打折扣。针对于此，镜头眩光效果常被限制在某些固定场合且无须刻意为之。类似的效果还包括光晕，此时，高亮区域边缘呈现为模糊状态，进而模拟相机的过度曝光效果。

旁跳镜头

当闪现其他场景时，旁跳镜头用于强调原场景中的某些元素。跳出镜头便是一例，且常用于显示某一动作的结果状态。例如，机械装置运转时一扇缓缓开启的门。该技术在游戏中亦较为常见，多用于展现玩家动作的结果，抑或亟待解决的下一个问题。

插入镜头

插入镜头类似于旁跳镜头，同样，当前场景在短时间内发生变化。其中，插入场景往往表现为原

场景中的某些细节内容。例如演员的近景画面（如头部特写镜头），进而凸显角色对某一事件的反应状态。

跳接镜头

与当前场景相比，跳接镜头是指针对类似场景所实施一种即时转换效果。这里，尽管两个场景间具有某些类似性，但后一场景往往具有不同的位置、不同的时间点（如白天和黑夜）抑或不同的阶段（如季节或岁月的变迁）。该方式在两个分离场景间提供了某种参考功能，以方便观察者对此加以评判。但在某些情况下，这一类镜头将会影响到视频剪辑操作，因而应尽量避免。

180° 规则/动作作用线

180° 规则和动作作用线分别采用不同的方式对两个角色间的作用线加以描述。在场景处理过程中，相机应位于该作用线的一侧，相机若跨越该作用线貌似会导致角色位置互换，如图 3.6 所示。

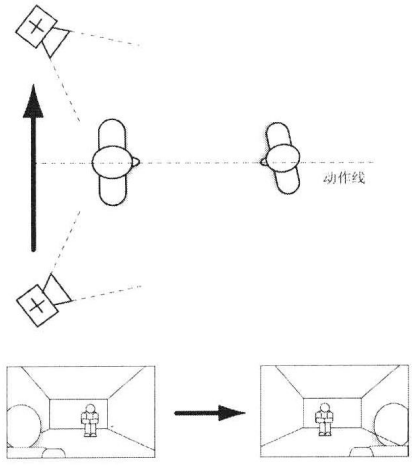


图 3.6 动作作用线

镜头视点

该术语表明，当前场景将从某一角色的视点（POV）处进行观察，这一点类似于游戏中的第一人称相机。需要说明的是，POV 镜头很少用于影视作品中，如果仔细加以处理，其高效性将会在游戏中发挥得淋漓尽致。

升降镜头

当采用升降镜头时，摄像机悬于吊臂上并从角色上方取景。在游戏中，这直接转变为第三人称相机，且在大多数情况下，相机将位于上述类似位置。有时，摄像机垂直方向上的移动也称作升降运动。

特写镜头

构建 POV 镜头或跟踪 POV 镜头以描述角色对事件的反应时，特写镜头将优于 POV 镜头。另外，特写镜头常用于拍摄角色聆听时的相关画面。

反转镜头

反转镜头是指当前摄像机镜头方向从上一个镜头处反转，但仍遵循 180° 原则。

30° 规则

当采用瞬时跳接镜头对当前相机实施重定位时，摄像机拍摄角度之间的差异应大于 30° （同一镜头内），小范围的移动通常会使观察者认为是常见对象在移动，而非摄像机相机自身。当然，对于上述行为，跳接镜头则是一个特例。

强迫透视

强迫透视技术常使得人眼产生错觉，并在物体的真实尺寸和视觉尺寸之间制造某种“误差”。某些影视作品中大量使用了强迫透视技术，并使得片中人物呈现出不同的尺寸。较为著名的例子是 Peter Jackson 所拍摄的“指环王三部曲”（Newline Cinema，2001—2003 年），片中角色均被重新修改，并针对其相对尺寸加以调整。最终，各角色可同时出现于屏幕画面中。

实时电影技术

在描述某些非交互性特征或对预录制视频进行回放时，大多数电影技术也适用于实时相机系统。类似地，第 1 章和第 4 章中的相关技术方案同样也适用于非交互式相机。亦即，可通过此类方案定义相机的行为和属性，如期望位置或移动方案。这里，常见的问题是如何选取适当的方案用以描述实时状态下的电影技术。关于电影相机序列，此处存在两种分类方案，其主要差异在于玩家是否可在上述序列中与游戏实现交互操作，下面将分别对此加以讨论。

电影

此处，电影显式地定义了一类具有非交互特征的视频序列，其中，玩家无法对所显示的画面信息进行任何修改操作。通常情况下，通过传统的胶片方式，视频序列事先拍摄完毕可实现稳定的播放以表现某一特定的事件。根据视频序列的真实表现方式，实时系统还可进一步划分为以下两种类型：

☑ 预渲染电影可视为一类非交互式视频序列并通过离线的方式进行设置和录制，影片的回放过

程将同步显示原胶片序列中所包含的内容。其中，单幅图像依次存储并按顺序加以播放。除此之外，电影画面还会采用相应的压缩技术以减少所需的数据量。同时，影片在拍摄过程中通常会采取相同的压缩率，比率间的差异可通过不同的国际电视标准（NTSC 制式或 PAL 制式）加以显现。

☑ 实时电影也可视为一类非交互式视频序列。尽管实际视频序列通常以离线的方式加以设置和录制，但实际内容将通过游戏引擎加以播放并对场景内的相关元素进行渲染。另外，视频序列可将预定义的相机行为（如静态相机序列）与动态生成数据进行结合，例如游戏对象的移动。该电影模式常作为一类精彩的视屏回放以示玩家，这在赛车类游戏中尤为常见。

实时电影的数据需求通常远远低于预渲染电影，当然，其潜在数据量亦不容小觑。必要时，游戏引擎还可引入动态元素，其中包括序列中玩家角色的定位（或合成）、动态视觉效果、调整渲染操作以反映角色的状态变化（如玩家生命值的消耗）等。因此，实时电影令游戏对象更接近于玩家体验，并生成独有（以及个性化）的视觉效果。

需要注意的是，非交互式电影与游戏的结合常会产生某些问题。当采用电影播放技术时，设计者需明确告知相关故事情节；同时，这将中断正常的游戏运行且玩家将从游戏画面中消失，因此，玩家在之前游戏体验中获得的临场感也随之消耗殆尽。当然，通过在游戏与视频间的实施精确的过渡转换，上述问题可在一定程度上得以抑制，例如可适当地移动相机并使其他特性与切换点实现良好的匹配。然而，一旦玩家失去了对角色的操控能力，即使上述过程采取无缝转换，全部过程依然会存在无法克服的间断感。此外，当交互特征消失时，应通过某种方式告知玩家以避免其处于茫然状态（如可调整相应的宽高比数据）。

针对于此，存在相关的替代方案以在播放视频序列时保持玩家与游戏之间的交互性，其形式亦多种多样并统称为交互式电影技术。

交互式电影技术

正常情况下，电影视频序列以及游戏之间并不包含交集，其原因在于：电影中的视频播放序列通常定义为非交互式，通过镜头的走位进而展现剧情的魅力以及丰富的画面信息等内容（在“可接受的”技术框架范围内）。电影作品中通常会弱化或消除玩家的交互特征，而游戏与电影视频相互结合的行为一般称作交互式电影技术。

根据玩家的游戏需求、游戏事件、玩家动作以及其他不可或缺的因素，交互式电影技术可动态地对画面内容加以调整。此外，视频序列可采用多种方式加以播放。大多数情况下，玩家并不能直接参与电影视频元素之间的互动过程，玩家只可采用统一的方式对其角色人物进行控制。有时，有必要对玩家的移动方式加以限制以确保电影视频序列元素处于可见（或听力）范围内。

针对交互式电影视频序列，较为常见的例子是 Boss 战，即在游戏结束之前与强大的敌方角色进行殊死搏斗。这种情况下通常需要对敌方角色的某些重要特征加以显示，如敌方较为明显的弱点区域。除此之外，有时还需要动态选定当前场景，在凸显其他游戏元素的同时以使玩家角色人物可见。

交互式电影技术可划分为以下 3 种类型：

（1）预定义交互式视频序列采用游戏引擎加以实时渲染，但相机的位置和方向则通过离线的方式加以定义。通常，相机的位置应经过适当选择以覆盖游戏可见区域，因此，与常规游戏画面相比，相

机需移至稍远处。另外，鉴于游戏引擎负责渲染视频序列，根据游戏需求，相机的方向应以动态的方式加以确定。

（2）实时交互式视频序列也可采用游戏引擎进行渲染，但其位置和方向则根据游戏以及电影视频动态加以确定。按照惯例，游戏元素应优先考虑，但在某些情况下，这将导致电影视屏内容不可见。许多时候，需要对场景环境加以精心设计并限制玩家角色人物的活动区域，进而确保视频元素的可见性。这里，较为著名的例子是游戏 Half-Life 2[HalfLife04]，其中，大量的视屏内容穿插于游戏中，且在视频播放过程中仍保持玩家的第一人称视点。同时，丰富的面部表情、非操控角色的交互特征以及精心安排的游戏脚本事件极大地丰富了玩家的游戏体验。

（3）预渲染交互式视频序列结合了预渲染图像以及实时渲染游戏元素。此时，背景元素可视为一类非交互式电影。这里，如果背景内容通过建模工具包或游戏引擎得以生成，其内容将（每帧）存储深度缓冲区信息，除了常见的图形数据之外，这也使得游戏对象更容易融入至 3D 背景元素中。当前，玩家的活动范围则被限制在预定义的路径之上，因此多少会使用到预渲染图像。原始数据还将进一步生成碰撞信息或深度信息，并进一步增强游戏场景的真实感。

在视频播放过程中，可采用多种形式以呈现玩家的交互特征，但其本质特征则较为有限，以防止局面过于复杂。[Glassner04]提供了大量的可选方案，其中较为常见的方法如下所示：

- ☑ 可根据玩家的选择方式采用相应的电影视频方案（如对话框模式或电影视频播放过程中基于游戏角色的其他交互方式）。
- ☑ 在电影视频播放过程中，可根据玩家的行为动作选择相关视频方案（如按下按钮或其他操控方式）。
- ☑ 根据前述游戏事件对电影视频进行调整（例如，关键人物的死亡、执行某种特定的动作等）。

更为重要的是，上述技术是否会增强玩家的游戏体验还需视具体情况而定。例如，在某些游戏类型中，玩家的视线将被阻挡，一些较为重要的游戏元素将无法得见，这将大大降低玩家的游戏体验。另外，某些启发型游戏无法针对各种情况提供相应的游戏视觉画面，因此，可能的话，应适当地保留玩家对相机的操控能力。即使如此，电影视频中的交互技术仍然提供了另一种思路，这也使得传统的游戏技术获得了长足的进步，同时也留有了更为广阔的拓展空间。

电影摄像机

电影摄像机普遍应用于当今的影视工业中且不会涉及任何交互特性。摄像机通过其走位和方向的控制向观众展示事物的特征。

摄像机的移动

帧间的连贯性是一类较为重要的特性，针对这一点而言，电影摄像机与相机之间的差异并不显著。然而，在非交互式视频序列播放过程中，相机的移动应降低至最低点。虽然相机的大幅度移动可能会产生令人激动的画面内容，但该项技术仍需谨慎使用。需要引起特别注意的是，静止的相机开始移动时，应确保缓慢加速；同样，相机停止之前，其减速过程也应保持均匀。在某些情况下，相机的急速

运动可用于强调场景元素的相似移动方式，但通常情况下，应尽量避免出现这一情况。同时，相机的高速运动可能会产生跳接镜头（具体内容可参见前述小节）。某些拍摄手法还会在摄像机的运动过程中添加晃动效果，该效果常出现于一些写实性电影作品中，其意图往往在于模拟手持摄像设备抑或增强观众的临场感，即通过削弱摄像机的镜头感以使观众达到一种身临其境的效果。不可否认，该理念值得称赞且在非交互式环境下切实可行。但在实时游戏中，为了减少相机行为对玩家动作的干扰，该技术应谨慎使用。

摄像机的方向

针对于重定向操作，电影摄像机也需要对其进行平滑的调整，但可能会对游戏体验以及玩家操控产生不良的影响。考虑到视频序列的非交互特征，与角色的取景并表现其丰富的情感世界相比，该操作所产生的问题不足以虑。

在非交互电影视频序列中，可摇动摄像机（即摄像机围绕其视见轴旋转）以获取相应的视觉画面，该操作十分有效并可增强场景画面的紧张感并制造一种混乱的气氛。然而，处于失控或快速摇动状态下的摄像机往往会使观众感到晕眩，可能的话，应谨慎或尽量避免使用这项技术。

动态生成画面/回放摄像机

针对游戏视频序列的回放操作，多数游戏均会提供这一功能或对此有所需求。同时，基于预渲染视频或预定义电影视频序列的实时渲染，这也可视为一项独立的功能。该方案通常称作回放相机，且需要对游戏元素以及游戏事件进行精准的复制并再次生成游戏视频序列。针对于此，回放相机应具备场景对象运动状态以及视觉状态的重现能力，并可通过下列方式加以实现，即每帧（或若干间隔帧）中对各对象的渲染状态加以复制，必要时还可采用插值计算以获取相对精准的结果。考虑到渲染排序操作将会增大内存空间的开销，因而可选择采用一类决策型方案。此时，仅游戏初始状态以及玩家输入状态得以存储。随着游戏过程的不断进行，上述存储值也将随之被替换。在同一时间间隔内，系统将对输入状态加以捕捉，并以此运行相关的游戏逻辑。当游戏结束后，相机将对已录制完毕的视频序列加以回放，这将大大降低内存的占用空间。

在某些情况下，回放相机可对前述相机视点实现精准的复制，该形式较为直观且在相机或其他游戏系统开发过程中，常被视为一类有效的调试工具。类似的方式也适用于其他游戏对象的视觉画面，如某些生物或车辆对象。再次强调，出于调试目的，该方式可调整（具有 AI 功能）敌方的视见内容进而对玩家的行为方式实现全方位的监视（以检测玩家的行为方式是否符合标准），游戏 Halo 2 [Halo04] 即采用了此项功能。[Isla05]对此进行了详细的描述。除此之外，上述方案还可向玩家提供一个与众不同的观察视角，例如，多玩家或编队游戏常会设置“旁观者”这一角色（亦即，这一类玩家并不参与当前游戏活动），并以玩家视角审视当前游戏的进程。因此，这也提供了一类潜在的学习工具，新晋玩家可通过此方式了解 AI 角色的决策方案以及战术部署。

回放相机的另一个功能是回顾近期所发生的游戏事件，该功能常出现于赛车游戏、飞行模拟以及飞行类游戏中。根据相机的定位、定向的动态程度，这一类系统则可视为视频播放技术的变化版本。

回放摄像机包含 3 种类型，即场景再现型摄像机、脚本型摄像机以及自动生成型摄像机。

再现型摄像机/关键帧摄像机

再现型摄像机采用一系列之前已存储的位置和方向数据，这一类数据在既定（或固定）时间间隔内生成。这里，存储数据的回放过程通常称作关键帧且可视为一种动画模式。实际上，三维建模软件包将采用与其他动画对象相同的方式生成关键帧。该方案可使设计人员在其较为熟悉的环境下工作，因而较为流行，但从本质上讲，该方案依然需要将游戏数据导入至建模软件中。

导出关键帧时摄像机数据的采样率可等同于游戏的更新频率，抑或采用稍低的采样率并通过插值计算以匹配于当前游戏的刷新率。再现型摄像机的具体实现过程则较为直观，且依赖于其他非关键帧游戏事件中的决策方案（如对象逻辑、碰撞反馈等）。

脚本型摄像机

脚本型回放相机可视为一类混合解决方案，且兼具交互性元素以及预定义元素。另外，设计人员往往会对回放相机的位置加以预定义。

- ☑ 不连续状态下的相机行为。基于不连续行为的脚本型相机将在两个相机位置间按照特定的顺序执行跳接转换或插值计算。这里，相机位置的变化条件包含各种因素，其中包括视频序列的时间、玩家角色人物的位置（绝对位置以及与当前相机之间的相对位置）以及脚本事件（如与时间相关的事件、敌方角色的死亡、建筑物的爆炸等）。

技术要点

相机数据的关键帧技术应事先最小化存储，相关数据包括相机的位置和方向、采样间的时间间隔、视域以及其他与渲染操作相关的数据。例如，如果数据的采样率定义为常数，则无须存储采样间的时间间隔值。另外，连续帧间的数据存储过程也并非一成不变。显然，还可使用标准的动画压缩技术以获得相应的渲染效果，如样条曲线过滤技术。然而，相机动画效果对于压缩操作十分敏感，与相机方向和位置相关的任何调整均会在渲染过程中有所反映。

- ☑ 基于路径的运动。基于路径的脚本型相机将遵循预定义路径实现其运动方式且该路径由游戏设计者加以定义。其中，相机沿路径方向上的位置数据还依赖于时间、玩家以及其他变量数据。另外，路径上的移动方式并非一成不变且该情况普遍存在。实际上，随着时间的推移，相机位置的非线性映射可实现自身的加速或减速运动，该方式常出现于电影的拍摄过程中。类似地，可简单地调整映射函数进而在相机沿预定义路径上移动时实现跳接镜头功能。
- ☑ 伺服运动。基于伺服运动的脚本型相机常相对于某一特定游戏对象（一般为游戏玩家角色人物）进行定位。同时，（一段时间内）相机可在相关位置间实现自动切换或经由玩家操控实现这一功能。针对该类型的回放型相机，较为常见的例子是赛车游戏，其中，可在车辆的不同部位实现场景画面的动态显示。类似地，脚本型视频相机的方向定位也可通过多种方式加以实现。
- ☑ 固定方向。在游戏体验过程中，一类较为常见的场景是采用固定方向强调特定的游戏事件、环境特征抑或表现某种感情色彩。当采用固定方向时，相机间的转换可通过一段时间内的、

方向间的插值计算得以实现，必要时，还可选择跳接镜头的方式。第6章和第10章分别对方向和插值问题进行了较为详细的讨论。

- ☑ 预定义方向。这里，可通过分段 Hermite 样条或其他形式的映射函数对相机采取直接控制，而非简单的方向插值计算——这也是 Hermite 曲线的用武之地。Hermite 曲线易于操控且常出现于专业的软件开发包中，如 Maya。在非交互式影视作品中，使用样条曲线控制相机的各项属性（或动画效果）十分常见。
- ☑ 对象跟踪。对象跟踪相机根据玩家或其他游戏对象的位置调整其自身的方向。除此之外，该类型相机还可在一段时间内对目标对象的位置进行预测。
- ☑ 对象取景。当某一对象（通常为玩家角色人物）需相对于显示设备出现于某一特定位置时，对象取景相机可调整其方向以满足这一要求。另外，通过修正相机的位置或视域范围，还可进一步修改目标对象的尺寸。同时，类似的技术还可用于调整相机的相关属性以确保多个角色人物同时出现于屏幕之上。当然，在某些情况下，这可能会导致相机的剧烈运动。

动态生成画面

回放相机的动态画面生成技术常出现于第三人称相机中，进而丰富视觉画面的显示内容，该技术常出现于赛车游戏的视见系统、动作冒险类游戏中玩家死亡时的画面显示（在某些情况下，还将通过不同角度对该场景进行重复回放）、格斗游戏中强调某种特殊攻击任务时的画面等。

针对回放相机的动态画面生成过程，其目标在于在实时渲染过程中获取多种移动以及定位方式，并辅以视屏操作进而获取不同角度的镜头方案。因此，即使同一视屏序列重复播放（某些游戏事件会重复出现），玩家也可从中获取不同程度的新鲜感。同时，个性鲜明的显示画面常会激发玩家更为强烈的兴趣。

动态画面回放型相机常会包含某些脚本机制并对定位选择方案提供某种支持。当然，针对某一特殊位置或场景空间，该方案也会对相机的定位进行某种限制。

结束画面

动态画面回放型相机还应具备播放结束画面的功能。当某一玩家任务失败后，相机应采取适当的方式表现这一场景（在多玩家游戏中，这也体现了对胜利者的一种肯定和褒奖）。有时，该方案可通过静态相机加以实现，当动画或特定的视觉效果序列播放完毕后，相机可聚焦于玩家角色人物。除此之外，其他形式还包括围绕玩家角色的移动，以及预定义位置或动态生成位置间的相机镜头切换。在某些游戏中，玩家角色人物任务失败时的游戏画面会重复播放数秒，并以不同的视角加以显示，这进一步增强了游戏的场景效果。针对于动态相机的生成位置，应确保相机位于当前场景环境内部空间中且不应与场景以及其他角色相交。在第一人称相机中，无须切换至第三人称相机视角以显示特定的结束画面。这里，可使相机指向地面以表明（或模拟）当前玩家角色处于死亡状态。当然，也可从外部视角针对玩家这一终结状态加以详细描述。随着视频内容的不断丰富，玩家的满意度也会随之提高，这一点在玩家被终结时表现得尤为明显。

开始画面

通常情况下，玩家进入游戏时（当前游戏进程存储完毕后，抑或在多玩家游戏中玩家角色被终结），相机往往会播放一段视频画面且这一类画面通常呈现为非交互特征。

有时，上述视频序列预定义为电影画面抑或引擎内的影视画面序列。然而，某些情况下，设计者根据游戏状态或出于向游戏添加视觉元素等目的（类似于前述结束画面），需要对上述视频序列进行调整。

回放相机的效果

通常，应向回放型相机添加相关属性并令玩家认为相机由某一模拟人物加以操控而非计算机。

重定向

这里，一类较为常见的实现方式是在相机的重定向过程中添加少许延迟。亦即，当对相机执行重定向操作时，可采取一定量的过度补偿（例如，可超出既定方向少许并于随后对该“错误”进行修正）。上述效果皆属于“非完美性”重定向方案，因此误差不可避免。针对于此，设计者应谨慎这一类视觉效果，亦即，误差量应处于可控范围内且不应出现于每一个镜头中。基于被跟踪对象的移动速度以及相机间距的启发型方案可用于调整上述误差量。有时，定位误差还将在脚本中加以注明并提示摄影师此处采用非常规手法，抑或标明相关物理条件将妨碍对目标对象的正常取景（如摄像机架设于颠簸水面上的船只甲板上）。

摄像机抖动效果

该状态是指相机处于物理运动状态下，进而表明摄像机受到某种外力的影响，如车辆运动或其他物理现象所导致的震动。当然，该现象也可能由相机装置的不稳定性造成（当摄影师行走于崎岖不平的地面之上尤是如此）。需要说明的是，这一类轻微震动常会导致捕捉画面模糊不清并会产生其他视觉副作用，所以应尽量避免。相机的抖动效果体现了轴向位置的偶然性，这种偶然性本质上可采用正弦曲线加以表述，其中包括移动相机时的正弦波振幅，以及确定移动速度的时间值。

这里，可向虚拟场景人为添加相机抖动效果。数学误差或移动干扰噪声所导致的偶然抖动往往不可取，这会分散玩家的注意力，同时，该现象也表明当前相机系统依然存在问题。另外，考虑到与玩家之间的动态交互特性（基于相机位置和方向），这一类视觉效果往往在实时环境下表现得十分明显。

鉴于当今手持摄像机均配置了防抖动功能，因而视频摄像机中的对应模拟技术已显得毫无必要。但是，该技术可向静态场景添加某些动态特征，并以此表明当前物理摄像机架设于场景某处。综上所述，相机的抖动效果确实应谨慎使用。当该效果与某一特殊的游戏事件结合使用时，其场景画面将给人留下深刻的印象，并可向玩家提供相应的反馈效果，进而增强玩家的游戏体验感。

在相机抖动功能的实现过程中，下列内容应引起读者的足够重视：相机的抖动效果通常在渲染过

程中执行，而非在游戏场景内对其进行移动，这将保留相机的真实位置且无须考虑抖动的幅度。尽管如此，设计人员仍然需要对抖动效果加以检测，以防止相机的渲染位置位于当前场景环境之外抑或视锥体与其他几何体相交。

需要注意的是，相机的抖动效果在第三人称相机中表现得并不明显。当采用第一人称视角时，该效果可向玩家提供丰富的视觉辅助元素；但针对于第三人称视角，该效果则会使玩家误认为当前相机系统存在某种问题。另外，相机抖动效果并不局限于非交互式视频序列，它也可应用于实时游戏中，并用于模拟爆炸效果、武器的反作用力、跌倒或跳跃动作后的震动效果等。

相机的摇动效果

在大多数游戏中，相机围绕其前向轴旋转并不常见，当小于90°角时，这一类旋转通常称作倾斜操作；当旋转角度大于90°时，则称之为镜头的摇动。在非交互式电影视频序列中，摇动镜头是一类常见的拍摄手法，并用于表现场景画面处于失控状态。在电影技术中，该操作也称作直摇或斜拍，且常采用固定的角度值。当从车辆的驾驶座舱中进行观察时，需要根据其机动性适当地摇动镜头，进而体现车辆处于颠簸状态。针对于第三人称视角，也可采用较少的旋转量以模拟车辆的运动状态。除此之外，镜头的摇动还可用于相机镜头间的转换操作进而强调物体的运动状态，当与其他图形效果结合使用时（如光圈转换），该方式尤为有效。

视域/景深

模拟相机镜头的不完美聚焦状态可视为另一种向电影摄像机添加真实感的手法，这可通过视域变化并结合景深效果得以实现，最终效果将对处于聚焦状态的对象实施稍许模糊操作。需要注意的是，模糊状态不应一直出现，当相机的焦点从一个物体移至另一个物体时，视域值应相对于目标值发生短暂的变化，与此同时，聚焦状态也应随之发生改变。另外，该操作还可用于对镜头中的无关对象进行模糊操作，进而将玩家的注意力转移至某一特定位置。

脚本机制

在一段时间内，拍摄者对相机的位置和方向进行定义，这可视为电影视频序列脚本中的主要内容。同时，脚本内容常在特定的工具中加以实现，例如场景编辑器，当视频序列通过游戏引擎进行渲染和更新时，这一点尤为明显。当拍摄过程受限于存储条件时，该方案提供了较大的灵活性，进而大大缩短了制作周期。另外，该方案还可通过游戏事件对其内部的视频序列加以控制，并通过相关游戏状态激活对应的游戏视频。除此之外，脚本机制还可对游戏对象实施跟踪操作，这对于动态生成的视频序列尤为有效。最后，该机制还可在电影视频序列与实时视频序列之间实施有效的匹配，进而获得统一的画面效果。

针对于此，一种处理方案是从建模软件包（如Maya）中读取相机以及其他相关数据信息，并由游戏引擎负责场景的渲染工作。该方案的难点之一是无法准确地复制建模软件包中的全部功能（如样条

曲线数据），与期望的结果相比，这将导致最终的视频序列出现少许变化。同时，游戏中的场景数据也应存储于建模软件包中。

如果不考虑脚本机制的具体实现过程，相机的脚本系统可提供多种方案以对视频序列加以操控，其中包括如下内容：

- ☑ 相机在一段时间内的位置和方向变化。
- ☑ 镜头变换。
- ☑ 相机间的插值计算。
- ☑ 视域变化。

针对于交互式以及电影序列，读者可参考第 6 章以获取对应示例。

电影脚本工具集的重要特征之一便是具备画面的快速播放以及预览功能，这也意味着，视频预览功能应内建于该工具集内，并可对相关变化进行同步察看（或许需要下载至目标机器上）。视频序列预览操作依赖于游戏逻辑以对场景元素进行操控，这也是实现该技术的众多难点之一。另外，预览工具还应支持当前游戏引擎，这进一步增加了预览处理过程的复杂度。针对于此，预览工具常出现于游戏中，而非编辑程序中。

多数脚本方案需要对相机的位置和方向进行显式的控制，其手法类似于建模软件工具包中的对应方案。某些方案还会使用到相应的场景描述语言，并需要进行手工输入。而较好的方案可通过某种方式对镜头类型以及相机位置加以定义。除此之外，系统还应根据镜头类型提供其初始位置，并随着预览过程的进行对这一类位置和/或方向数据进行调整。

一类较具特色且与电影技术相关的应用程序可根据动态游戏场景生成对应的视频序列，亦即，这一类程序可通过某种方式以表达期望中的情感特征（或与当前游戏氛围相匹配），并根据动态变化的开始和结束条件消除游戏环境以及视频序列之间的界限。当然，可对该方案稍作变化以使相关设计人员以较为统一的方式确定视频序列的最终表现意图，但为了实现这一目标，依然需要通过计算机设备来解决相机的定位以及运动等问题。虽然这一类自控方案切实可行，但最终结果仍有可能处于不确定状态。在很大程度上，这将取决于初始条件以及视频的表现意图。当然，某些较为简单的情形即使在动态环境下其处理手法也会相对直观。

与设计人员探讨视频序列自控操作通常会引起他们的警觉，该操作会替代其工作内容或削弱相关人员在团队中的地位——此类想法过于保守，实际上，视频的自动处理过程仅是一类设计工具，且可加快设计人员对事物的认知过程，并提供更为丰富的移动以及方向类型。因此，其实现过程存在一定的必要性，并可进一步完善相机的脚本机制。

自 20 世纪 80 年代起，自动化视频技术一直是研发领域的热点所在，相关论文对此也进行了较为详细的阐述。例如，[Hornung03]和[Hawkins02]即对此内容分别引入了不同的解决方案。

剪辑操作

剪辑操作可视为影视技术中的一项重要任务，该操作从美学观点出发并结合某些既定规则以生成最终的图像序列。这里，剪辑操作的基本构成内容是镜头，即一系列连续的、录制完毕的图像集。镜头的排列和拼接构成了剪辑操作的基本任务，当然，该任务通常在影视作品拍摄完毕后执行。另外，

实时剪辑操作包含以下3个重要元素，即镜头选取、取景以及场景转换。

镜头选取

某些颇具代表性的镜头体现了强烈的感情色彩，针对于此，游戏中的镜头选取方案与主流电影技术并无太大差异。在某些方面，游戏镜头的选择方式则更具灵活性，相机角度和位置可包含多种可能性，进而可满足各类镜头要求。然而，电影导演则通常会受限于此（这也是采取多角度拍摄的原因）。

取景/合成操作

关于摄像机的定位和方向，标准电影技术拥有一套特定的规范以表现相应的情感状态，抑或强调某些特定的情节元素——该规程定义为镜头且构成了全部电影技术的基础元素。这里，镜头的主要内容取决于取景操作，镜头类型一经确定后，需要对其主体内容以一种较为特殊的方式执行取景操作。根据电影技术中的相关规则，取景操作可视为镜头选择方案的具体实施过程。另外，根据对象在屏幕取景器中的定位方式，相机与其拍摄对象的同一相对位置可呈现出彼此不同的情感状态。

上述技术中的大部分内容源于传统绘画技术以及剧院的舞台设计。实际上，早期的影视作品即采用与传统舞台技术相同的方式加以展现，其中，静态摄像机采用固定的方向进行拍摄。随着时间的推移，该过程增加了演员角色的表演内容，从而进一步丰富了舞台效果。同时，角色的距离和方向处于不断变化中。此处，镜头合成技术可视为电影技术中的重要组成部分，即屏幕取景器中物体的排列方式。[Arijon91]对此进行了深入的讨论，其中两条较为重要的规则是作用线（前述内容已对此有所描述）以及称之为三分构图法的合成方案。这里，三分构图法定义为一种屏幕划分方案，场景中较为重要的元素将被适当地加以定位，以使最终的拍摄结果符合审美要求。必要时，可绘制平行于屏幕水平轴以及垂直轴的直线，以此将屏幕划分为尺寸均等的9个区域，其中，较为重要的元素应置于直线的交点或附近处，如图3.7所示。除此之外，垂直或水平元素应与某一直线对齐，与仅将目标对象置于屏幕中心位置相比，该方式将生成更具意义的合成结果。需要注意的是，在交互式环境下，上述规则仍需对当前游戏需求加以考察。

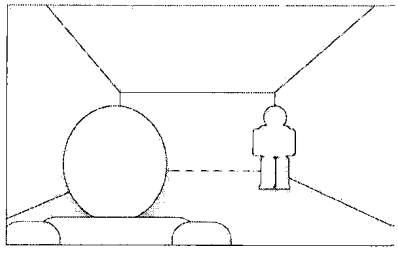


图3.7 三分构图法

场景转换

场景转换通常定义为某一场景至另一场景之间的切换，针对传统的电影技术，当涉及屏幕角色的

定位时，理解该转换方式并不困难。通常情况下，转换操作往往会包含两个场景中的公共元素。这将对观察者提供一个公共参考点，即使相关公共元素并非完全相同。例如，当主要角色在两个场景间稳定行进时，该角色首先走入镜头（此刻，视线完全被遮挡）并于随后走出镜头且场景呈现为完全不同的状态。当在转换过程中维护相机的运动方向时，同样会发生类似的变换。此时，随着视线被前方物体（如树木）遮挡，场景也相应地发生变化。

下面将对常见的基本转换形式进行简要的阐述。

跳转镜头

在全部转换操作中，镜头跳转可视为最为简单的一种方案。在当前游戏场景与另一完全不同的场景之间，镜头跳转可实现二者之间的快速转换。这里，视线变化应大于 30° 角并确保场景变化时镜头确实发生跳转，而非相机自身的跳动。

需要注意的是，场景元素均可实现跳转镜头切换。然而，某些既定技术规范表明，应在两个场景间留有若干公共元素（如角色人物的相对位置或方向数据）。

淡入/淡出转换

淡入/淡出效果是一类令人赏心悦目的变换操作，但其计算开销也不容小视。淡入/淡出效果需要在两个场景间执行 Alpha 混合计算并执行渲染操作。如果渲染单一场景时其性能已然捉襟见肘，则该效果的实施将面临很大的考验。

当然，也存在相对简化的方式实现上述效果，亦即，可首先获取源场景帧（通常情况下不包含非场景与渲染元素，如玩家分值数据、装饰图层等）并于随后在该捕捉屏幕画面与新近渲染的结果场景之间实施 Alpha 混合操作。另外，其他效果也可应用于捕捉后的源场景画面之上，例如可将其分解为三角形或其他 2D 多面体形状，并对原场景重新加以整合以形成新场景，抑或将原场景置于某一模型表面之上等。

圈入/圈出转换

在动画中，圈入/圈出是一类较为常见的场景转换方式，亦即，以同心圆的方式从屏幕中心向其边缘更新当前画面。这一类转场方式常用于清除屏幕画面抑或显示某一全新场景。较为常见的情况是，连续使用该方式清除画面并在不同位置对画面内容进行更新。圈入/圈出转换类似于视口转换，具体内容稍后解释。

划接

划接操作可将屏幕清除至某一固定颜色，抑或用于显示某一新场景。针对于前者，屏幕画面常被清除为黑色。此时，画面将被连续移动的黑色线条填充，其水平或垂直移动方向通常平行于显示设备的边框。在某些情况下，划接变换还可在对角线方向或采用圆形方式加以实现。更为常见的方式是采

用数字合成技术以非线性方式对屏幕画面进行清除，当然，实际情况还要视具体的场景元素而定（如汽车挡风玻璃上的雨刷）。

在电影画面中，上述划接方式常与逆划接结合使用，并以与原划接操作相同的方向生成新的场景画面。通过该方式，最新场景画面貌似位于原场景画面之后加以显示。另外，划接操作还可与相机的移动（摇动镜头）结合使用。需要说明的是，若对两个不同场景进行划接转换，其计算开销亦不容小视，该操作类似于上述淡入/淡出转换，亦需要对两个场景加以渲染。然而，单一场景可视为静态场景，这将大大降低处理器的需求；同时，划接线条可视为场景中的唯一内容，因而对当前场景实施像素操作即可，这一点与全屏淡入/淡出效果还是有所区别的。

视口转换

视口转换体现了转换操作的不同形式，即该转换方式可动态地改变渲染窗口的实际尺寸。例如，在电影放映过程中，屏幕可被临时划分为多个区域，每个单独区域分别显示不同的场景画面或人物画面。这里，该划分效果可即刻发生。通常情况下，多个视觉画面的连接则可通过平滑的过渡加以执行。例如，可放大某一视觉画面直至其填充全部平面，或通过某种有趣的方式对窗口的尺寸或形状实施动画（或插值）操作。

工具

针对游戏视频序列的创建过程，是否存在对应的工具集用于定义和操控视频相机的属性以及上述转换操作亦是众多关键因素之一。相关制作人员应对标准电影技术以及建模或动画软件包所提供的制作工具了然于胸，并依此生成对应的场景。针对完全采用预渲染的电影视频序列，上述软件包应提供相应的技术支持。另外，若场景通过游戏引擎加以渲染，则建模软件包还应提供最终的场景数据信息以及视频脚本支持，进而可对常规游戏对象加以定义和操控。

然而，针对于预渲染操作，游戏与视频之间的无缝转换需要对预渲染场景及其对应组件中多边形的复杂度加以限制，进而实现二者之间的良好匹配。除此之外，针对某些游戏中难以实现的特殊渲染视效，则有必要减少其使用频率。例如某些光照视觉效果，实时游戏通常无法对软件渲染方案所包含的同一类型的真实感提供良好的支持。

如果视频序列采用游戏引擎加以渲染，抑或当视频序列中包含游戏环境或游戏对象变得十分重要时，场景编辑器应提供相关功能项以实现对视频序列的有效控制。某些开发环境则基于专用的开发工具以满足特定的实时游戏需求；除此之外，针对特定的建模以及动画软件包，一些开发环境则通过插件的方式满足上述要求。抛开具体的编辑环境不谈，提供一组公共功能集对于视频序列的正确实现是十分必要的，相关内容如下：

- ☑ 一段时间内，基于显式操控的相机的位置和方向变化。
- ☑ 基于另一游戏对象属性的相机位置和方向变化（例如，跟踪某一对象或自动调整方向以朝向某一对象、响应另一对象的运动并移动当前相机）。
- ☑ 针对不同相机间的变化实施操控，除了标准的跳转镜头之外，该过程通常还包括插值计算。

- ☑ 根据视见或相机的位置激活相应的游戏事件（如音频效果、音乐、粒子系统等）。
- ☑ 在不同的视频序列之间进行转换（如前述的划接操作）。
- ☑ 根据流逝的时间值、相机位置以及其他相关因素，对视频相机的视域加以控制。
- ☑ 根据前视方向控制相机的方位，如摇动镜头。
- ☑ 关卡编辑工具应提供用户可控的、视频相机行为的预览功能，进而减少设计过程中的时间消耗。其中，预览内容应准确地反映原视频序列中的运动方式、方向以及其他相机属性。

另外，出于编辑以及调试目的，编辑环境以及游戏内部应可从外部视角对视频相机进行检测。通常情况下，相关属性可实现为时间段与某一范围值之间的映射操作。虽然线性映射简单实用，但某些较为灵活的系统还会提供分段式 Hermite 样条以及三次多项式曲线，进而可对属性的变化率提供更为精准的控制。图 3.8 显示了基于视域映射的相关示例。

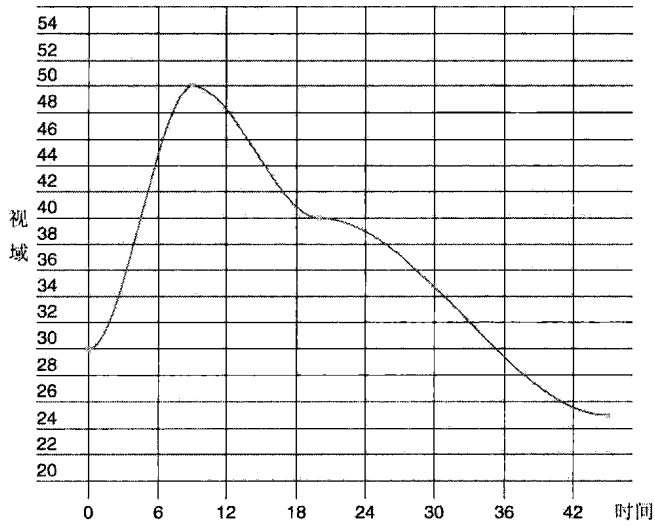


图 3.8　分段式 Hermite 曲线可在相机属性（如视域）与时间段之间实施有效的映射

视频序列的预览功能十分重要并可有效地降低设计过程中的时间消耗。在跨平台开发环境的编辑工具中运行游戏逻辑通常不具备可行性，该情况往往需要根据当前开发系统实施远程通信，进而获取（下载）编辑环境所做出的调整性内容。理想情况下，上述变化貌似发生于本地系统，且兼具动态调整视频序列时间值的能力，因此可在本地开发系统上直接察看相机的移动以及方向状态。

本章小结

理解并掌握标准电影技术可向读者提供全方位的帮助。首先，针对于游戏设计者以及对于实时应用程序不甚了解的艺术从业者，它构筑了二者之间的一条有效通道；其次，该技术所包含的诸多规则同样适用于交互式环境，并提醒开发人员游戏中的审美需求同样不可或缺。另外，该技术的重要性还体现于视频游戏的开发过程中，并以一种易于理解的方式展现故事情节以及人物特征。

第 2 部分：设计原理

当读者初步了解了虚拟相机的基础概念及其与实时应用程序之间的作用方式，即可进一步考察相机行为类型与特定环境之间的工作方式。游戏体验的分析过程以及相机的选取需要读者深入理解潜在的可选方案。另外，相关知识将有助于读者设计新型的相机行为，并对现有相机创建对应的变化版本。通常情况下，基于现有行为的简单变化可生成更为有趣的替代方案。针对于此，本部分内容将针对这一类决策方案提供必要的信息。

第4章 相机设计

游戏相机的设计过程需要对游戏需求有着较为透彻的理解，并辅以相应的美学特征。本章将主要讨论相机系统设计过程中的基本概念，其中包括游戏场景的显示方法，以及如何将这一类方法应用于特定的游戏类型中。除此之外，作为游戏产品中的重要组成部分，本章还将考察真实相机系统的设计过程及其设计需求。

第 2 章曾讨论了游戏场景的显示方法，当通过相机系统加以显示时，相关方法常称作表现风格。游戏相机通常会采用某一种表现风格，尽管大多数游戏在单一表现风格中包含多种相机行为。从广义上讲，相机行为可通过其与场景玩家的相对位置加以分类，亦即第一人称相机和第三人称相机。另外，相机行为（也称作相机类型）还将包含大量的特定属性（例如期望位置的确定），但通常情况下，可将其简单地定义为第一人称相机和第三人称相机。

当然，可在相关游戏类型中使用特定的表现风格以及行为并对此进行多方尝试。全新的透视关系可大大提升玩家的视觉感受以及游戏体验。在交互式序列中，相机行为的首选方案即是令玩家获取最佳观察视角，这一点与具体的游戏类型并无太大关联。然而，当采用相关游戏设计方案或场景细节作为表现手法时，玩家应对相机拥有控制权。

游戏相机的行为还应考虑到玩家人物的尺寸和移动属性，以及玩家与场景环境和其他游戏对象之间的交互方式（如有效范围内的交互操作、近距离战斗等）。在某些情况下，相机行为的选取方案往往难以确定，但相关设计哲学依然需要与特定的游戏环境相适应，相关内容将在本章后续内容、第 5 章以及第 6 章加以阐述。

另外，本章还将探讨基于同一输出设备的多视点显示方案，这一类方案常出现于多玩家游戏中（单一玩家游戏偶尔也会采取多视点显示方案）。

如前所述，游戏的表现风格主要分为两类，即 2D 和 3D。另外，针对 2D 或 3D 场景中的背景以及游戏对象，还可采取相应的混合方案。在后续内容中将会看到，每一种表现风格均包含一组公共功能集。除此之外，上述表现风格还适用于交互式以及非交互式（或电影）视频序列，下面将分别对每一种表现风格所涉及的交互式和非交互式特征加以考察，以进一步理解二者之间的差异。

交互式 2D 相机系统

这里，可采用与 3D 相机系统相同的方式（不包括正交投影）考察 2D 相机系统，不难发现，二者之间存在着较大的差异。需要说明的是，出于历史原因，2D 方案受限于硬件的计算能力（如早期的游戏机游戏、手持设备或移动电话设备），因此相机或游戏设计者需要对其表现风格做出某种妥协。虽然 3D 加速硬件（以及高速处理器）技术获得了长足的进步，但某些游戏依然会采用 2D 方案以彰显其简洁风格和画面处理的独特性。另外，大量 3D 游戏还会包含一系列附带小游戏，这一类游戏通常会采用

较为简单的 2D 表现风格。

早期的 2D 游戏并不会涉及相机这一概念，这一类游戏只是简单地将游戏场景的位置数据映射至输出设备之上（通过离屏缓冲机制），且该映射机制可通过多种方式加以实现。其中，较为常见的技术是贴图映射——再次强调，这一类技术源于当时的硬件条件并可在一定程度上节省内存开销（或降低场景世界所占用的带宽）。贴图映射通常采用二维数组加以存储，并使用贴图索引值获取相应的数据。这里，每一个索引值均代表了一组小型图像集（简称为贴图），各图像经适当拼贴后以形成一幅较大的图案。最终，显示设备所呈现出的映射结果呈现为一个窗口，且该窗口沿贴图表面移动。鉴于贴图中的各元素将以索引值或哈希值的方式加以存储，因此，重复使用同一贴图元素时将会大大节省内存的开销。

技术要点

需要说明的是，存在多种方式以实现贴图元素的内存存储。例如，无须对各贴图元素实施单独存储，针对于此，可采用稍大的图像以节省内存空间。当然，其对应的索引机制以及渲染方案的复杂度也会随之提高。另外，贴图集可通过显式的世界坐标定义其相对坐标空间，而非贴图集内的隐式位置。当贴图集内的元素较为稀疏时，该方案颇为有效且可为设计人员提供较大的灵活性，尽管在处理碰撞检测问题时该方案可能会稍显复杂。通过此方式，需要一个屏幕缓冲区以实现贴图元素的定位操作并对其实施渲染。某些硬件配置仅可存储某种格式的贴图，即该贴图需对齐至某一规则边界。当然，贴图元素的尺寸并无固定限制，因而可采用各种二维图像以生成游戏的场景世界。

另外，贴图还可视为一系列的图层数据且每一贴图对应于相应的权值。如果贴图通过图层的方式加以组织，则画面将呈现出一种伪三维效果。根据贴图图层的相对权值，该方式使得游戏人物可在图层间移动。另外，贴图图层无须实现彼此对齐，因此，通过在轴向上对图层进行适当的偏移，即可实现视差效果以及多图层滚动效果。

贴图还可用于标记需更新的显示区域，当硬件设备不完善时，该方式则较为常见。同时，主处理器负责生成显示内容。这里，无须对全部贴图执行更新操作，贴图中的部分区域将被标记为“脏”数据或记为“需更新”，如精灵的绘制过程。此时，仅将被标记的区域复制至显示设备中。该技术多见于早期的 8 位计算机设备中，其图形硬件加速条件十分有限。

期望特征

2D 相机系统的主要特征之一便是确保玩家人物一直位于屏幕之上，在某些情况下这将会产生问题并导致游戏状态发生变化，如单屏游戏或单视口多玩家游戏。2D 相机系统特征集包含大量的、特定于当前表现风格的元素。多数情况下，针对玩家角色以及场景位置，这一类元素均与相机的位置相关且称之为滚屏操作，即背景图案相对于相机产生移动效果。当然，实际情况则是相机位置相对于背景进行移动。针对于 3D 相机而言，期望位置的确定以及相机的移动方案直接反映了玩家的感知状态。通常情况下，滚屏的距离以及方向往往取决于玩家角色人物的移动；另外，滚屏操作亦可独立于玩家的移动并呈现为连续的滚动状态；在某些情况下，滚屏操作还可与特定的游戏视见发生关联（例如，当出现 Boss 战时，屏幕将停止滚动；当战争以胜利而告终时，屏幕将恢复至滚动状态）。

下面将对常见的滚屏操作进行逐一分析。

连续滚屏

连续滚屏是指在游戏体验过程中，背景呈连续滚动状态，其速度和方向将根据具体的游戏需求而定。这里，速率和方向并非一成不变，进而可防止玩家接触到屏幕边缘。但通常情况下，玩家的位置对滚屏的速度和方向无明显影响。在某些游戏中，如果玩家不主动向滚屏方向相反反向移动，则滚屏

操作将会顺势带动玩家人物移动。如果玩家被强制移出当前屏幕范围内（例如，如果滚屏速度大于玩家自身速度，抑或鉴于环境特征，玩家被迫受阻），则该玩家将处于死亡状态。

通常情况下，玩家的位置将被限制在某一轴向上（也有可能限制在两个轴向上，具体内容参见下一小节）。对于垂直滚屏游戏，玩家相对于屏幕具有固定的垂直位置，当然，其水平方向上的移动也是允许的。一般而言，玩家角色人物仅可在垂直于滚屏方向上移动，其限制范围往往还涉及包围盒。此处，包围盒的各面分别平行/垂直于滚屏方向，进而在一定范围内限制玩家的移动。

基于玩家的滚屏

根据需要，相机的运动在一个或两个轴向上与玩家的移动同步进行（即不存在延迟）。然而，鉴于人物角色移动所导致的画面改变，双轴通常难以取得令人满意的效果，根据玩家人物的移动特征以及控制设备产生的输入噪声，这往往会导致画面的抖动。尽管如此，玩家角色与显示设备之间的相对位置则不会发生改变。多数情况下，该位置为屏幕的中心位置，但在某些滚屏游戏中，尤其是滚屏方向保持固定时，相关角色将定位于显示设备的边界处，进而可获取较为开阔的视觉画面。

方向延迟

相机的期望位置常取决于玩家的移动方向，且相机与期望位置之间的移动过程并非即时完成。因此，玩家与显示设备之间的相对位置亦不再固定。需要注意的是，玩家应确保位于安全帧内（参见第2章中的相关内容），为了达到这一要求，滚屏速度应与玩家的移动速度匹配，或适当地大于玩家的移动速度。

画面的突进

相对于当前相机，其期望位置将保持固定，直至玩家到达期望位置（或行驶了一段特定距离）。此时，随着时间的推移，相机将自动调整至某一新位置，期间，玩家的控制操作将暂时失效。这将导致游戏场景的新增画面以突进的方式移动，在新增画面的显示过程中，鉴于玩家需要在游戏场景中保持同一位置，因而玩家的屏幕位置将发生变化。多数时候，根据滚屏方向，玩家的屏幕定位通常会返回某一标准位置。

基于屏幕的滚屏

当玩家移至可见区域的边缘处或屏幕内预定义的边界处，则相机开始启动。一旦玩家沿某一方向移动并企图脱离当前屏幕范围，相机即处于连续运动状态中。

相机的移动速度取决于玩家的速度，因此应确保玩家保持相同的屏幕相对位置，直至位移方向发生改变。当采用预定义边界时，玩家通常被限制在屏幕区域的较小范围内，如基于中心位置约25%屏幕范围内。

基于区域的滚屏

当玩家跨越场景边界时，相机即处于启动状态。在一段时间内，相机持续移动至期望位置处。此处，期望位置保持固定不变，直至玩家跨越另一场景边界，该过程将会一直重复执行。

确定视见内容

显示于输出设备上的2D贴图区域有时也称作视窗或视口。在早期的2D游戏中，硬件往往通过视口实现贴图的轴向对齐。当采用软件渲染方案（假设处理器的性能足够强大），或使用纹理渲染硬件加

速技术时，该需求则变得微不足道。另外，当与多重贴图图层机制结合使用时，还可进一步实现旋转、基于视差的多层背景效果。

当采用 2D 表现风格时，游戏相机的焦点无须直接位于玩家角色人物之上。此处需要强调的是，相机的位置可独立于角色的位置加以定义；另外，还需考察玩家角色人物与场景环境之间的交互方式。如果玩家角色人物在较长距离范围内实施瞄准或其他交互操作，则有必要对该角色进行预测。由于人物角色可实现快速的重定向操作，因此相机观察位置的确定则变得更加复杂，具体内容将在第 7 章加以详细讨论，其中一点值得读者留意：针对 2D 相机系统，相机的观察位置与期望位置之间应实现对齐排列（鉴于正交投影）。

技术要点

2D 相机系统与其对应的 3D 相机系统之间具有诸多共性，例如，可针对同一系统在某一运动轴上加以限制。相机系统经过适当抽象化后，其底层相机对象甚至并不会了解具体的投影方式（正交投影抑或透视投影），因此，视口操作（插值计算、淡入/淡出等）可采用同一规范加以处理。

在基于视口的多玩家游戏中，游戏相机的位置往往取决于玩家的位置。针对于多玩家游戏，该位置常等同于全部玩家位置的平均值或玩家包围盒的中心位置。在某些情况下，该位置还可通过各玩家的权值加以计算，进而确定最终的期望位置。另外，某一特定角色的权值可根据游戏的具体环境加以定义（如 Boss 战）。

针对玩家的运动行为，此处依然存在某些限制条件。例如，若玩家行进至某一位置且根据两玩家间的互见性，该位置将阻止玩家移向其他位置。最终，两玩家皆无法移动。

不难发现，针对于潜在的动态游戏效果，上述方案难以令人满意。此处的限制条件是，玩家角色人物需位于屏幕范围内，如果不向玩家提供角色人物的控制操作，则上述要求很难实现。一种可能的解决方案是将视口划分为两个独立区域，且分别对应于各自的玩家。若该划分方案确实可行，这将向玩家提供全新的体验。另外，该方案甚至允许玩家实现离屏移动——当然，这并非上策，且这一类情形常定义为“不存在玩家移动”（有时也称作“软锁”）。如果视口可实现缩放效果，则可移动相机并使其远离背景平面且进一步缩小当前画面。当然，该情形也可通过玩家一方的适当移动得以实现，但无法确保该状况一直处于正确状态。更为复杂的方法是将玩家“瞬移”至某一有效位置。

针对于大多数 2D 相机系统，需要将视口限制在贴图范围内，显示游戏场景的外部区域通常不可取，这将破坏游戏画面的完整性。有时，还需要对视口基于游戏地图的相对位置加以适当的限制，以阻止玩家观察到场景中的部分内容。这里，可将贴图元素标记为“不可见”并在确定相机新位置时对潜在可见贴图元素进行测试。

2D 电影相机系统

2D 电影相机系统以图层方式对二维图像进行合成，进而生成其显示画面。其中，可通过调整图层的绘制顺序、添加透明度抑或不规则的图形形状以获取画面的深度感。这种表现方式类似于早期的卡通以及漫画技术。例如，可采用该技术模拟相机“进入”场景时的运动方式，亦即，相机附近处的场景元素沿屏幕中心位置向外移动，类似的效果还可用于静态照片之上，进而可生成更为有趣的幻灯片

效果。[Thomas95]对该表现风格提供了较为权威的指南。

2D电影相机系统具备以下显著特征：

- ☑ 视口摇摄控制。针对于场景地图的可见视口，该控制操作定义了相机的移动路径。通常情况下，这一类路径可设置为一系列的路点抑或曲线路径。另外，在一段时间内，相机还应对其位置加以控制并以非线性方式移动（从而可较为方便地靠近/远离目标位置）。
- ☑ 相机的缩放功能。这里，可相对于2D元素移动相机，进而模拟其缩放效果（抑或对视口或其内部元素实施缩放操作）。另外，缩放效果还可通过适当地移动视口内的图层得以实现。
- ☑ 对象跟踪功能。相机的移动方式无须通过路径或既定位置加以显式定义，但却可通过与目标对象（通常为玩家）间的相对位置加以定义，尽管该相对位移在一段时间内可能会发生变化，抑或受到其他拍摄环境的限制。
- ☑ 将对象置于屏幕边界内或其他基于视口的约束范围内。

交互式3D相机系统

在全部相机处理方案中，交互式3D相机系统则相对复杂，其实现过程也颇具挑战性。此处，设计任务的难点在于，相机在满足游戏动态变化的同时还需获得精致的画面效果。

需要注意的是，交互式相机系统应向观察者提供基于正确场景关系的游戏画面，这也意味着游戏中的美学观念不可或缺，至少不应被忽视。在适当的时候，相机系统应针对相机的方向或位置提供相应的控制操作，在某些情况下，该操控过程亦会出现问题（如针对某些几何约束条件）或与游戏设计的期望结果背道而驰（如暴露隐藏元素或渲染后的图像包含某种缺陷），最终导致控制操作失效。通过统一的界面以及视觉元素，玩家有权利获取与相机操控相关的信息。

通常，3D相机系统拥有庞大的特征集，并构成了全部电影视频相机的环境需求。当然，这其中可能会包含玩家动作行为所导致的相关问题。因此，灵活的相机系统使得设计人员可选取较好的解决方案并大大地提升玩家的游戏体验。

常见的相机系统特征包含下列内容：

- ☑ 可选取相应的投影渲染类型（如透视投影或正交投影等）。
- ☑ 可方便地确定相机的位置（抑或提供漫游功能）。
- ☑ 可充分体现相机的运动特征，即相机如何从当前位置移动至期望位置。
- ☑ 可确定相机的期望方向。
- ☑ 可充分体现相机的重定向特征，即相机如何从当前方向转变至期望方向。
- ☑ 针对相机属性的渲染操作，可对其实施动态调整，如视域和宽高比。
- ☑ 可提供相机位置和方向的插值计算方案。
- ☑ 当相机无法移至有效位置或玩家角色人物不可见时，系统应提供相应的保护机制。
- ☑ 可提供单一显示设备上的多视口渲染操作（即分屏）。实际上，该功能与投影方案的选取方式无关，因此等同于2D相机。

相机投影转换将改变对象或几何体坐标在游戏场景中的表现形式，并最终体现于渲染输出设备之上。同时，渲染机制（无论是软件渲染或硬件渲染）通过投影变换将世界坐标转换至显示设备坐标。在

某些情况下，该转换操作仅用于计算屏幕空间位置（如基于预定义图形元素的正交投影或正等轴测投影）。在大多数 3D 投影中，上述变换常用于顶点坐标的转换操作进而执行多边形（或更为高级的对象表面）的渲染操作。另外，最终的转换过程通常会得到硬件的支持，但游戏系统需提供相应的矩阵数据。

这里，3D 相机系统包含两种常见的投影类型，即透视投影和平行投影。

透视投影

透视投影多出现于 3D 游戏中，某些 2D 游戏也通过对象的缩放或视差效果对此加以模拟。由于透视投影多用于模拟人眼的立体视觉，因此确保该投影机制的正确工作还是十分必要的。从本质上讲，透视投影实现了场景位置的映射过程，即反向跟踪场景位置至视点之间的某一直线，其中，视平面与量位置间的直线相交。此处，平行直线最终会交于某一消失点，因此，远离于相机的各个对象其尺寸将相应地减小。大多数计算机图形学专著均对此进行了描述，如[Akenine-Möller02]或[Foley90]。

平行投影

游戏中的平行投影呈现为多种类型且具有共同的属性，即源于对象上的平行直线经视见变换后所保持的最终形式（并以此加以分类）。

正交投影

正交投影常见于工程绘图中，通常情况下，正交投影多采用 3 视图表达法（即俯视图、正视图以及左视图）且各视图之间均围绕自身对象中心位置旋转 90°，并以此方式实现视见对象的轴对齐。正交投影保留了直线间的平行关系且各轴之间具有相同的缩放比例。除此之外，当采用正交投影投影时，对象间的相对比例亦保持不变且无须考虑其与相机间的相对位置。

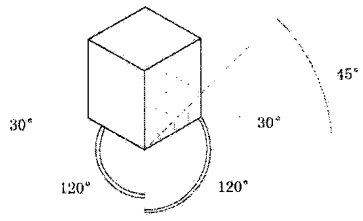
由于采用了非透视方式且投影方向平行于某一投影轴，因此正交投影多出现于 2D 游戏中，针对于仅显示贴图数据的游戏机硬件，这可视为一类首选方案。不仅如此，正交投影也常用于 3D 游戏以向玩家提供与游戏相关的辅助信息，如显示于屏幕某处的场景地图、雷达显示信息等。在 2D 游戏中，正交视见元素可通过预定义的图像加以生成，抑或采用相关转换矩阵进行渲染。

轴测投影

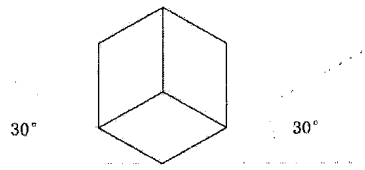
轴测投影隶属正交投影的一个子类，其投影方向无须与视见对象轴对齐。在轴测投影中，对象的 Up 轴通常保持垂直方向，即使根据所选用的轴测类型其他两个轴呈现为不同程度的倾斜状态。轴测投影包含 3 种类型，即等轴测投影、二测投影和三测投影，而游戏中较为常见的类型则是等轴测投影。

- ☑ 等轴测投影。严格地讲，等轴测投影包含 3 个等间隔轴（即各轴之间的夹角为 120°）且各轴向上均包含相同的缩放系数。另外，后向轴的缩放系数约为垂直轴缩放系数的 80%。这里，缩放结果可通过下方 45° 直线（平行于垂直轴）上的投影距离获得，如图 4.1 所示。出于计算的简单性，游戏中侧轴上水平步进尺寸与垂直步进尺寸之间的比率常定义为 2:1，因此，与水平方向上的夹角约为 26.56°，而非 30°。考虑到该角度值已成为事实上的“标准”且十分接近于实际角度值（该值更便于计算屏幕位置），因而已被人们广为接受。等轴测投影更为符合人类的审美标准，因此，采用该方式而非预渲染或绘制后的 2D 元素渲染多边形数据则变得

顺理成章。需要说明的是，正交投影可分别于对象的上方或下方处加以绘制并显示相应的视见内容。通常情况下，游戏中的相机视角多位于地形上方，当然，令相机提供 4 个不同的场景视角（分别对应于各主轴方向）并无不可，但与预渲染的图像相比，这将占用大量的开发时间和/或存储空间。虽然可通过镜像方式适当地降低内存开销，但最终的效果依然无法逃脱玩家的法眼（例如，图像翻转后，角色人物右手中的武器瞬间位于左手中）。



（a）从上方进行观察时的等轴测投影



（b）从下方进行观察时的等轴测投影

图 4.1 等轴测投影

- ☑ 二测投影。在二测投影中，两个坐标轴具有相同的比例以及角度（相对于第三个坐标轴），如图 4.2 所示。需要注意的是，二测投影与等轴测投影之间的差别在于，3 轴之间的夹角相等。因此，二测投影适用于轴测示图，其真实感稍逊于三测投影。另外，二测投影常用于早期绘画作品中，与文艺复兴时期的透视投影技巧相比，其历史更为悠久。
- ☑ 三测投影。三测投影则对投影轴间的角度比例不做任何限制。类似地，在保持平行直线的同时，轴间的比例可随意变化。与其他类型的平行投影相比，三测投影则更具真实感，如图 4.3 所示。

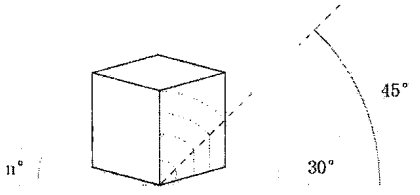


图 4.2 二测投影

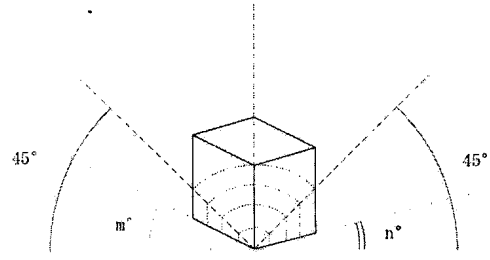


图 4.3 三测投影

斜测投影

斜测投影较少出现于游戏中，尽管该投影方式支持基于贴图而定图形方案。斜测投影中的两个轴平行于投影平面；第三轴则相对于水平方向后倾45° 或60° 角。斜测投影包含两种主要形式，即 Cavalier 斜测投影和 Cabinet 斜测投影，如图 4.4 所示。Cavalier 斜测投影在全部 3 个轴向上均使用相同的比例值；而 Cabinet 斜测投影则在后倾轴上使用将比例值减半。斜测投影则可提供更为真实的正面视觉效果，亦即，画面将呈现为符合比例尺寸的宽高比数据。



（a）Cavalier 斜测投影　　（b）Cabinet 斜测投影

图 4.4　斜测投影包含的两种主要形式

3D 电影视频相机系统

视频系统应涵盖传统电影中所涉及的各项技术，这里，假设游戏引擎负责渲染视频序列，至少开始时定义为此方式。后面将会发现，该方式包含了诸多优点。另外，视频相机系统还应提供真实相机设备所不具备的某些功能。

在设计 3D 视频相机系统时，读者有必要了解视频序列的生成方式。针对于建模软件包中的视频生成操作，美工师往往对此轻车熟路，因此，了解传统电影技术中的既定规则将有助于实现与美工人员之间的顺利沟通。下列内容包含了 3D 视频相机系统开发过程中的注意事项：

- ☑ 复制建模软件包中的控制项以及功能项。可能的话，应采用与建模软件包所提供的、相同的操控方案。有时，相关数据可从软件包中提取；而在某些情况下，视频序列则可在场景编辑器中加以定义，这将大大简化相机系统的开发过程并进一步提升设计人员的工作效率。针对于此，加强与美工师之间的交流是十分必要的，这将使相机系统的功能项更具针对性。
- ☑ 采用通用技术开发相机系统。在条件允许的情况下，应尽可能地使用标准电影视频技术。当设计人员描述并使用标准视频设备时，其表达方式将更为清晰。另外，如果相机系统在缺乏对其移动和定向实施直接控制的条件下依然可创建场景镜头，则可根据相应的电影术语方式表现镜头的构建规则（如特写镜头、头部拍摄镜头、双人特写镜头等）。
- ☑ 考察预渲染电影视频序列。出于性能考量，该操作将大有裨益且在视频序列播放过程中生成动态视觉效果时效果显著。其中，视频序列渲染时可能会使用到游戏引擎无法生成的某些特殊效果，抑或仅采用较为常见的渲染操作，随后，该视频序列将以压缩格式加以存储并在必要时以内存流的方式加以显示。不难发现，渲染过程以及视频序列的压缩过程将十分耗时；除此之外，压缩视频还将会在游戏传输媒介上占用大量的存储空间。当然，视频序列也可通过游戏引擎加以渲染，该方案将在一定程度上降低存储需求并缩短开发周期；同时，视频序列可以一种更为自然的方式与当前游戏渲染风格融合于一处，进而提升游戏画面的整体效果。
- ☑ 在游戏中对非交互式视频序列加以区分。当视频序列与游戏之间实现无缝结合时，其画面效果往往令人赏心悦目，但这也存在着潜在的问题。此处，应确保玩家能够意识到已丧失了对相关角色的控制权。某些视觉线索可用于强调相应的视频序列播放操作，例如视口宽高比的改变（标准的电影宽高比比值为 1.85:1），这对于 SDTV 播放格式而言并无不妥且变化效果相

对明显；然而，对于 HDTV 宽屏显示设备，其宽高比变化并不明显，除非采用某些较为极致的调整方案（例如，当宽高比为 2.35:1 时，画面将产生较为严重的变形）。

- ☑ 相机系统应允许玩家提前退出。一般而言，初始画面开始播放时，玩家应有权利选择跳过该视频序列。该过程可能会占用大量的时间，因此玩家宁愿选择直接进入主游戏界面，该情形在玩家重启游戏时尤是如此（也就是说，玩家有可能被迫多次观看同一视频）。

如果不考虑显示画面的投影方式，大多数视频序列将与对象的移动、动画、图形或声音效果以及其他游戏事件（沿单一时间轴）实现同步操作，因此，应提供相关工具并随着时间的推移对相机属性进行适当的调整。通常情况下，这一类属性常包含运动方式、方向以及视域等内容。

关于电影技术的深入讨论，读者可参考第 3 章以获取更多内容。另外，第 6 章还阐述了电影视频序列创建过程中的脚本技术。

2.5D 相机系统

游戏场景的混合表现方法通常结合了 2D 和 3D 技术，进而可生成独特的游戏场景画面（如游戏“红侠乔伊[Viewtiful03]”）。然而，该方案并非“真实”场景画面，但却提供了诸多选择以及游戏技巧。另外，这一类游戏还可使非游戏玩家快速熟悉当前游戏环境。许多休闲类游戏制作者或玩家通常对二维显示表面上的三维投影倍感陌生，针对于此，3D 可视化界面往往成为其步入三维游戏世界的重大阻碍。如果相机系统无法对游戏画面进行有效的表达，这一类玩家往往会就此放弃游戏的体验过程。通过对游戏场景加以适当的限制，可使玩家的注意力集中于游戏元素之上，而非其表现风格。当然，该方案并非万全之策，但某些 3D 游戏可将其作为一种借鉴进而增强玩家对游戏的体验感。

游戏设备

针对于渲染游戏场景的显示设备，虽然相机系统无须了解其细节功能，但宽高比数据依然不可忽视——当涉及多种显示格式时，这一点尤为重要。例如，宽屏格式或全帧显示格式。除了调整宽高比之外，运行于个人计算机之上的游戏通常还应支持多种分辨率格式（即有效物理像素的数量）。

大多数家庭视频游戏设计为仅支持单一显示设备，通常情况下为显示器或电视设备；手持游戏设备一般会提供内嵌显示屏幕。另外，多数计算机游戏以及街机游戏均支持多种输出设备。针对于前者，只需在操作系统与当前应用程序窗口之间实施较为简单的映射即可。当然，这并不能保证最终显示结果的宽高比以及分辨率保持不变。而对于街机游戏，显示机制自身即是游戏设计中的一个重要组成部分，一个较为明显的例子是 Nintendo DS 手持游戏系统中多显示屏的使用，其中，两个显示屏以垂直排列的方式加以布局。除此之外，某些游戏还在主显示设备（如电视设备）和手持游戏设备之间实现分离输出，即独立的游戏机（如任天堂公司所研制的掌上游戏机）或者内嵌于操控设备或内存卡中的显示设备（如置于世嘉游戏机中的 VMU 内存卡）。

宽高比

某些情况下，显示设备的宽高比与游戏自身的宽高比并不匹配。在第 2 章中曾简要地介绍了 4 种较为常见的宽高比格式，其中两种直接对应于电视显示标准，即 SDTV（4:3）和 HDTV（16:9）。计算机显示器所提供的宽屏格式（分辨率为 1920×1200，宽高比为 16:10）通常与上述标准并不匹配。

当游戏采用特定的宽高比以适应某种不同的显示格式时，其匹配方案亦多种多样。这里，一些处理方案并不会对游戏体验产生显著的影响；而某些方案则会导致游戏丧失原有的真实感。另外，早期显示设备通常难以支持高分辨率显示画面，进而无法获取较为精致的细节内容。因此，在游戏设计初期，上述因素需逐一加以考察。为了实现游戏输出画面与各类显示设备之间的良好匹配，下面将对相关处理方案加以简要的介绍。

调整尺寸

如果内存和硬件条件允许的话，可采用一类较为简单的方案使得渲染缓冲区和视口与当前显示设备的输出分辨率和宽高比之间实现良好的匹配。许多个人计算机均提供了多种分辨率选择方案，以满足不同的处理器性能要求。

镜头的摇动和扫描

镜头的摇动和扫描过程类似于顾客挑选 DVD 时的情形，电影的全帧版本可从原始宽屏版本中获得。此时，原始版本将被适当地加以剪裁（即移除操作），则剩余部分将满足宽高比要求。然而，该操作也意味着原始图像将损失 30%的内容，同时，在电影播放过程中，这也会显著地影响场景的观感。需要注意的是，原始宽屏格式的图像可采用 4:3 格式加以显示，虽然这将占用较少的物理空间，但分辨率也会随之下降。

拉伸效果

某些显示设备具备拉伸画面的能力，进而可实现宽高比的匹配操作。另外，针对用户自定义的形状以及尺寸，基于视窗操作系统的个人计算机设备也具备相应的画面拉伸能力，并确保宽高比数据得到当前游戏的有效支持，以最大程度地保持原始画面的完整性。

相机设计

相机设计结合了美学理论以及相关游戏需求，其开发过程往往比较耗时。针对于大多数设计技巧而言，对现有系统进行考察和分析可视为最为重要的实践元素之一。当面临某一款新游戏时，可通过考察（解构）当前相机系统的方式对玩家角色加以操控，并进一步观察相机系统针对特殊情况的处理

方式，以及相机基于角色运动、方向变化的行为方式。

当然，作为游戏设计的一个组成部分，无须过分强调相机设计的重要性，玩家相对于游戏的观感很大程度上取决于游戏的表现方式。因此，有必要将相机设计作为游戏设计过程中的一个基础模块加以考察。自然地，相机设计则是初始设计任务中的重要组成部分，其对游戏的持续性影响将会贯穿于全部开发过程。这里，其他游戏系统可能还需要某些特定的开发资源，除了相关开发工具之外，它们常用于游戏相机系统的设计、实现以及优化操作。当然，这也需要设计人员具备相机行为的设计以及操控能力，并进一步理解相机系统与游戏体验之间的关系。另外，相机的设计工作应在开发过程的前期完成，这样，即可在初始阶段发现相机系统中的某些潜在问题并及时地加以调整。考虑到相机设计结果将直接影响到场景环境，因此，可先对游戏场景进行调整，并于随后制订模型以及纹理的修改方案。

游戏以及技术需求

初始状态下，应对游戏需求进行适当的评估，并于随后针对场景的布局、各类素材等内容制订相应的规划方案。通常情况下，与美学需求相比，游戏需求分析具有较高的优先权，但二者均会受限于当前技术条件。在许多情况下，美学需求常会与场景内的游戏需求产生重合，即游戏体验常依赖于视觉满意度以及游戏场景中画面的可接受程度，但这并不意味着可完全照搬对应的电影技术规则。

设计概览

作为开发过程中的重要组成部分，下面将讨论相机设计的具体操作内容。这里的问题是，如何定义相机设计过程中的相关步骤？相机系统与游戏中的其他设计内容如何实现有效的关联？针对于相机系统中的相关功能项，可采用一类自顶向下的设计方案，如下所示：

- ☑ 考察高层设计目标。
- ☑ 对玩家角色人物的行为进行评估。
- ☑ 确定场景环境范围。
- ☑ 定义基本的相机行为。
- ☑ 确定特定区域范围内的场景需求。
- ☑ 相机系统的技术评估。

下面将对上述各步骤进行详细的讨论。

考察高层设计目标

相机设计的首要步骤即是理解相机系统的需求内容，即确定游戏的整体规划及其对相机设计的影响方式。例如，如何确定游戏不同组件之间的共性？单一表现风格是否足够？如何定义最终的表现风格？相关问题还包括：与游戏设计人员探讨游戏的表现方式，以获取最佳显示风格；考察类似环境下既定的相机处理方案，以及特定环境下相机系统的期望属性。

评估玩家角色行为

玩家角色人物的运动方式以及与场景间的交互方式将直接影响相机的设计方案。这里，如果设计人员对玩家角色的行为缺乏足够的了解，将难以生成正确的游戏体验。例如，玩家角色人物以较慢的速度行进时，其行为与飞行器相比将存在显著的差异。然而，当玩家的行为随时间的推移而发生改变时，相应的相机方案是否有必要亦随之发生变化？例如，玩家在游戏场景中获取了新的行进方式，如交通工具、跳跃能力、飞行能力等。

确定场景环境范围

游戏相机应在当前场景范围内充分发挥其各功能项，并根据相机的定位以及运动方式考察当前环境设计的隐含功能。例如，与峡谷相比，封闭的地牢则会与相机发生更多的接触。另外，玩家是否可在两种截然不同的场景间实现自由穿梭？

为了确保当前场景满足游戏体验以及相机需求，相机行为的先期原型至关重要，一旦游戏区域（有时也称作室内空间）构建完毕，即使该场景结构仅是一类雏形，也依然有必要对相机的具体行为加以考察——至少，几何体的复杂度以及相机定位之间的关系应相对清晰。另外，几何体的预期变化以及场景布局应在相关美工元素之前先期加以确定。

定义基本的相机行为

当上述 3 项内容确定无误后，即可定义基本的相机行为。亦即，除了后续调整过程中所导致的细化操作之外，上述内容在很大程度上确定了相机行为的基调。当然，针对某些特殊场合，还需采用与之对应的特定方案。例如，可对相机的定位以及方向实施精准的控制，进而强调某些特定的游戏元素。因此，应快速地构建相机原型以实现基本的相机行为。鉴于游戏体验主要取决于相机的表现方式，因而相机方案的确定通常具有最高的优先级。但不可否认的是，简单的方案往往更易于实现。

特定区域范围内的场景需求

当相机对某些特殊情况无法实现正确处理时，应对各场景环境加以多方分析。其中，针对各个区域，有必要从场景环境以及游戏元素角度对该区域中的细节内容加以考察。例如，玩家的移动被限制在某一封闭隧道中，此时，应从远距离、高架位置处对该隧道加以观察。

首先，可对玩家的移动行为的细节内容加以分析。这里，相机在移动过程中需要进行相应的定位；同样，具备一定 AI 功能的地方角色也需要适当地加以定位。上述操作应在环境布局、添加各类艺术素材等内容之前进行。通常情况下，与美学需求相比，游戏需求分析具有较高的优先权，但二者均会受限于当前技术条件。在许多情况下，美学需求常会与场景内的游戏需求产生重合，即游戏体验常依赖于视觉满意度以及游戏场景中画面的可接受程度，但这并不意味着可完全照搬对应的电影技术规则。

相机系统的技术评估

当确定最终的游戏方案时，基于相机系统的处理器以及内存需求依然需要得到足够的重视，在这一点上，任何游戏系统均无例外。如果特定的游戏环境需要某种自定义类型的相机方案，则应从软件工程以及技术需求的角度对此加以考察。通常情况下，可对游戏体验需求以及当前场景环境进行小幅度调整，进而可避免使用软件工程方案。这里，最为简单有效的问题处理方法即是设计正确的方案，从而使问题远离当前设计方案。

因此，这里需要对相机系统的性能进行持续性评估，例如，可采用多部相机对场景实施同步渲染。

技术限制

关于相机系统的技术限制，可分别从以下4个方面加以考虑。

（1）处理器性能

当对场景环境进行测试或实现其漫游功能时，相机系统的计算开销不容小视。例如，游戏场景中的投射光线以及其他空间分析过程。此处，可通过预计算方案适当地降低上述计算开销。某些动态系统每帧将会投射多条光线，通常，可对此采用分摊机制实施相应的优化操作。另外，还可通过省略某些材质以限制相应的测试内容，或者添加特定于当前相机系统且已经优化的碰撞几何体。

（2）内存性能

通常情况下，相机系统常需要基于场景环境的辅助信息以调整相机行为、消除碰撞或玩家遮挡效果、漫游基于复杂几何体的场景世界等。需要注意的是，根据场景的复杂度以及细节程度，单独的碰撞数据信息将会占用大量的内存空间。

（3）脚本功能

根据效果游戏需求，对于设计者而言，如何界定调整相机行为的控制程度？另外，某些辅助功能是否有必要实现手工编码？数据驱动方案提供了较为灵活的方式，但其维护成本相对高昂且学习曲线也较为漫长。第6章将深入探讨脚本技术。

（4）调试功能

理解相机行为背后所隐藏的内容对于相机系统的有效控制至关重要。这里，是否可采用一种易于理解的方式揭示相机系统的内部工作机制？是否存在相关方法用于操控自由态的相机并对位于相机系统外部的游戏事件进行观察？读者可参考第11章以获取更多信息。

相机设计中的问题

相机系统及其行为历经一段时间的设计后，应对游戏设计的评估过程建立合理的标准。针对特定的游戏设计需求，可创建相应的问题列表并制定相应的策略，以在游戏设计开始之前实现对问题的全面理解。

玩家行为以及控制

玩家的操控方式及其在游戏引擎内部的行为能力与相机设计紧密相关，具体内容如下：

- ☑ 玩家以何种方式移动？是否加速缓慢？玩家的移动方式是否被限制在某一路径或表面之上？玩家是否具备跳跃或飞行能力？
- ☑ 玩家的移动或行为能力是否为最终方案？需要注意的是，修改上述两项内容将会极大地影响

到相机的需求环境。

- ☑ 相机是否与玩家人物合而为一？
- ☑ 根据角色人物的方向，是否有必要保留某一位置以显示特定方向上的视野？
- ☑ 玩家视野中是否存在较为重要的元素？
- ☑ 相机应聚焦于玩家角色自身之上抑或是前方一点处(针对配备了瞄准装置或武器装备的游戏)？
- ☑ 玩家是否存在某一部位一直处于可见状态（如脚部）？
- ☑ 是否需要构建其他游戏对象而非玩家角色人物？
- ☑ 根据玩家的位置或其他游戏状态，相机的方向是否需要进行调整？
- ☑ 玩家角色人物采用何种控制方式？
- ☑ 是否需要使用与当前游戏体验不同的操控方案？
- ☑ 操控方案是否会根据当前游戏体验需求发生变化？

场景环境问题

针对于相机的可能位置，如果缺乏预测运动路径或扩展性分析（以及存储），则该相机很可能无法有效地处理包含任意复杂度的游戏场景环境。相机系统应能够实现动态的环境漫游并显示相应的场景画面，且满足美学要求以及游戏体验需求。某些场景环境的空间特性则无法满足上述定位条件，这一类情况应尽可能地加以避免，进而在特定的场景环境方案投入使用之前，构建良好的相机定位以及移动模型。相反，在艺术方案确定之后再对相机设计方案进行调整将对设计人员的心理产生极大的伤害，其工作量亦会随之增加。可能的话，设计人员应围绕当前游戏体验以及相机限制条件对场景环境展开设计。当然，在设计过程中，相关的折中方案也是十分常见的。

针对于特定的游戏环境，当考察相机系统的工作方式时，设计人员应注意以下相关事项：

- ☑ 角色人物的移动环境如何？场景世界是否局限于某一狭小空间内，抑或是相对开放的空间进而可以方便地实现相机的定位以及移动？
- ☑ 场景环境是否影响相机的定位或定向？例如，角色人物可能会穿越水面，针对于此，相机应避免与近剪裁面产生相交。与开放表面上的飞行动作相比，封闭、狭小空间内的相机处理方案则截然不同。
- ☑ 相机是否需要定位于场景之外？若是，则如何表现玩家角色？相机如何在二者之间实现正常转换？
- ☑ 相机是否需要跟随玩家穿越动态变化的场景环境？若相机已被禁止运动，玩家是否可穿越当前相机？
- ☑ 相机是否需要躲避较为复杂的几何体或大型的游戏对象？
- ☑ 相机的运动是否需要定位于特定的路径或对象表面之上，进而实现有效的躲避？
- ☑ 几何体是否存在变化进而简化相机的设计与实现方案？
- ☑ 相机是否存在较为特殊的操作需求？例如，除了玩家角色人物之外，可对某一特定的游戏对象加以显示。
- ☑ 哪种表现手法最佳？第一人称相机抑或第三人称相机？玩家是否具备不同的行为进而生成各类相机操作？
- ☑ 如何实现不同相机行为间的变化操作？
- ☑ 针对于既定场景环境，是否可确保相应的插值计算得以顺利执行？

- ☑ 当玩家采用不同方式进入场景时，如玩家从不同位置进入U型房间时，是否存在不同的相机处理方案？
- ☑ 是否需要构造相应的碰撞几何体以针对相机添加标记？
- ☑ 通常情况下，碰撞几何体一般采用斜角状对象，进而可实现相机在对象表面间的平滑移动。因此，碰撞几何体通常有别于渲染几何体。

技术问题

除了前述章节中所探讨的常见问题之外，还需对与相机系统紧密相关的游戏引擎的性能加以考察，具体内容如下：

- ☑ 当设计相机的移动、定位以及方向时，是否需要考察相关的技术限制条件？此处，这一类限制条件通常包括渲染时的性能问题、游戏人物以及几何体的近似度、控制参考帧的确定方案等。
- ☑ 如果相机必须穿越某些游戏对象或几何体，是否需要对其渲染操作进行适当的调整以使显示画面更为和谐？例如相关对象呈现为透明状态。
- ☑ 是否有必要阻止相机移动至其期望位置处？若是，如何对此实施进一步处理？
- ☑ 是否有必要在相机系统中内置自动保护机制以处理正常游戏体验中的突发情况？例如，位于相机和目标对象之间的门缓缓慢关闭、相机置于对其游戏场景外部、相机与其目标对象相互穿越等。
- ☑ 相机是否可实现重定位操作？该操作到达何种程度才可以令人接受？
- ☑ 是否可对相机加以定位以使游戏对象无法经过其位置抑或阻挡玩家角色人物？否则，应采取何种方式控制对象的渲染操作以满足上述条件？
- ☑ 针对于无法处理的玩家角色，当前环境是否能够提供相应的遮挡机制？另外，当前环境是否可适当地加以调整或重新构建，并在必要时被移除？
- ☑ 是否可在不同相机之间进行插值操作或实现平滑的转换操作？若是，转换过程如何处理阻止该操作的某些几何体或对象？另外，如何在不同相机行为或属性之间进行插值操作？

上述问题列表并非全面，但对于相机设计的初始阶段，这已然提供了足够的信息。此处，建议创建相关文档以描述相机系统的性能以及在上述约束条件的限制下游戏设计的工作方式。该文档无论对关卡设计者抑或相机设计者都起到了良好的提示作用，进而使关卡的设计过程充分考虑到相应的技术限制条件；另外，针对特定游戏的相机解决方案，这将有助于设计过程达到期望的美学效果。

制作流程

针对于游戏中其他结果数据的制作过程，相机系统有必要融入其对应的设计规则。针对于此，某些较为重要的设计阶段形成了相应的制作流程。作为一名相机系统的设计人员，了解该流程中的各个阶段及其对应的操作步骤是十分必要的。

游戏开发具备高难度、高开销、周期长等特征，因此，根据相机系统中的既定要求，设计人员有必要在全部开发过程中对其保持清醒的认识。同时，基于相机系统的限制条件应在项目早期即确定完毕并具备一定的健壮性。通常情况下，相机系统往往无法包罗万象，因而应具备添加辅助功能项的能力。针对于大多素游戏系统而言，应谨慎对待项目后期所出现的任何修改操作。当游戏系统相对简单时，其后期调整工作或许不会产生较为严重的后果甚至无伤大雅，例如修改一扇门开启时的时间值。

在大多数情况下，在项目后期对游戏体验需求进行调整往往会带来致命的灾难。针对不同模块间

的制作人员，良好的沟通、交流则是必不可少的。

素材需求

一旦相机系统的重要特征及其用途确定完毕后，即可对游戏设计中所涉及的相关元素搜索对应的素材（针对较大规模的项目）。在实际操作过程中，应设置一名设计人员负责相关任务的管理工作，其责任包括游戏表现风格的设计、定义并实现多数游戏体验中所使用的基础型相机方案。

该管理人员针对相机系统设计执行有效的监控并在各部分间提供单一接口以及接触点。然而，鉴于游戏设计的复杂性，管理人员的数量也呈现出上升的趋势。在主设计师的管理之下，多名设计师经协同工作进而完成相机脚本机制以及游戏相机的其他各类用途的设计工作。

需要注意的是，针对游戏的表现方式以及相机对游戏操控以及玩家感知的影响方式，相机设计均需要达到一定的美学要求。实际上，该技能随开发过程的不断进行将逐步得到完善。在相关人员的指导下，相信其他设计人员也会对相机问题有着更为全面的认识。同时，根据玩家的满意度以及体验程度，针对相机设计的全面理解可视为游戏设计的基本组成部分。

游戏设计原则

作为游戏设计的重要组成部分，相机设计哲学与游戏设计紧密相关且需引起足够的重视。同时，高质量相机系统还取决于丰富的素材资源以及足够的开发时间——但这一切究竟如何实现？

在某些情况下，相机设计依赖于设计者的个人经验并与游戏类型以及游戏体验需求紧密相关。一些设计选择方案取决于游戏的表现风格；而另一些方案则与游戏类型有关（如运动类游戏与冒险类游戏之间的差异）。尽管如此，设计过程中依然存在某些通用规则且与游戏类型无关。某些设计原则依赖于表现风格；而另一些则在很大程度上取决于游戏的设计需求。虽然存在某些特殊情况，但大致原则基本如此。需要注意的是，从玩家角度来看，最终实现方案应具备实际意义，例如，某些原则仅适用于第三人称表现风格，因此应引起足够的重视。

- ☑ 应确保玩家位于可见范围内。几何体或其他游戏对象（抑或视锥体剪裁操作）的全遮挡效果将会使玩家失去原有的方向感。然而，大多数游戏对此并未引起足够的重视。注意，这并不意味着全部观察点均聚焦于当前玩家角色人物；另外，部分遮挡以及临时遮挡依然允许。相机视见位置的确定及其与玩家之间的关系将在第 6 章中加以详细的讨论。
- ☑ 应禁止相机穿越（或靠近）游戏对象或当前场景环境。如果相机视锥体近剪裁面与渲染几何体之间相交，其渲染结果往往难以预料，这将大大降低游戏的图像效果且无法获得专业人士的首肯。在相机系统中，该问题将令人难以接受，但其解决方法并不复杂。这里，可简单地向问题几何体添加透明效果，进而避免上述现象的发生。另外，可适当地移除相关几何体（在实际应用过程中，可根据相机的近似度执行该操作），以使相机实现自由的穿越且不会产生任何副作用。
- ☑ 除去特定的设计需求之外，应禁止玩家对相机进行无谓的操控。针对第三人称相机而言，即使在最为复杂的情况下，相机系统也应具备自动选择适宜方案的能力。当然，在绝对必要或有意为之时，玩家对相机进行操控还是允许的。但通常情况下，则无此必要。针对于相机的操控，许多游戏对此持放任的态度。这里，如果玩家被赋予相机的操控能力，则应对此加以

谨慎处理以确保玩家不会将相机置于场景外部；同时，还应避免相机被错误地置于某一位置进而导致玩家被遮挡。相机的移动范围以及控制方案应与当前游戏设计以及技术需求实现良好的匹配，但第一人称相机则是一个例外，其中，相机的控制操作可视为正常游戏体验中的一个重要组成部分。即使如此，相机在某一时间段内也可实现自动调控以辅助玩家的游戏体验且不会产生任何冲突。通常，在定义明晰的情况下，上述方案对玩家的辅助效果并不明显。当相机沿倾斜面上行或下行时，可适当地调整垂直方向上的倾斜角以辅助相机的瞄准功能以及场景的漫游功能。

- ☑ 针对于特定的游戏设计需求，可对相机操控适当地放松限制要求。在某些游戏场合下，相机的控制操作将被禁止，但一味地否定玩家的控制权力有时看来则过于严厉。自然状态下，显示某一场景画面无须进行额外的控制操作，但也存在某些特殊情况，例如，设计方案可能会要求玩家对场景进行细致的检查；尽管玩家的企图通常难以判断，但相机的操控机制应可在较大程度上满足玩家的控制要求以及决策要求。基于玩家的当前状态以及期望动作，无法对场景实施有效的观测无疑会令玩家倍感迷茫。针对于该问题，相机设计者难逃其责。另外，若未加以充分解释，启动相机的控制功能后即对其加以限制将会令玩家无所适从。
- ☑ 尽可能地减少相机的无谓运动，针对电影视频播放序列尤是如此。这里，应尽量避免相机的无谓运动，除非特意追求相机落后于玩家角色人物或二者之间彼此穿越这一类效果。另外，还应消除玩家对场景环境的反馈（抑或源于玩家输入设备中的噪声）所导致的轻微移动。类似地，如果相机位移直接作用于玩家角色上，场景画面则会显得过于“僵硬”且十分不自然。
- ☑ 确保相机以平滑的方式运动。当相机以平滑、帧间连贯的方式运动时，则可最大程度地保持玩家的注意力以及定向操作。通常情况下，平滑运动可通过向速度数据添加阻尼这一方式加以实现。鉴于相机平滑运动的重要性，当玩家加速运动时，相关技术可使得相机处于稳定状态。另外，低通过滤器可有效地消除运动过程中的不规则状态，当相机运动与玩家角色人物直接关联时尤是如此。其中，玩家输入中所包含的噪声将导致相机发生轻微的震动。
- ☑ 限制相机的重定向速度。除非相机处于玩家的直接操控下，否则，相机快速或不连续的重定位操作将使得玩家失去方向感。鉴于相机的朝向决定了场景的绘制内容，因而相机的重定位操作使得全部渲染画面执行重绘操作。通过限制重定向的速度，该效果将降低至最低点。在第三人称相机中，这将导致短时期内目标对象不可见。当显式执行场景跳转并辅以相机的重定位操作时，即时重定向操作还是允许的，但新方向需要维持一段时间以使玩家适应当前环境。另外，玩家控制参考帧可有效地改善上述问题（具体内容参见第2章）。然而，即时重定向操作的出现频率不应过于密集。这里，建议采用较小的夹角值进行重定位操作。
- ☑ 在大多数游戏相机中，应允许适度的旋转操作。通常情况下，旋转操作将围绕相机的前向轴进行——再次强调，这将导致游戏处于不稳定状态；但在视频序列播放过程中，该操作还是允许的。偶尔，旋转操作可作为插值序列加以使用，进而调整玩家视野中的场景画面，抑或作为特定的游戏体验模型（如沿墙面爬行或匍匐前进）。某些游戏则故意采用该效果进而使玩家失去方向感，但在大多数情况下，该操作应谨慎使用。针对于上述规则，飞行模拟器则是一个例外。通常，飞行模拟器从驾驶视角展现游戏场景的视觉画面。即使如此，大多数人对过度旋转操作依然会措手不及，因此，将倾斜角度限制在一定范围内不失为明智的选择。有时，可按照某一固定角度进行旋转操作，进而强调游戏体验之间的差异性，抑或侧重描写玩家的感情状态。另外，在视频播放系列中，旋转操作则无过多限制。

- ☑ 应防止相机移动至游戏场景外部空间。在大多数情况下，相机应置于场景几何体内部，玩家无权查看游戏环境的构造方式，进而避免对游戏视觉体验造成不良影响。当然，例外情况是存在的，此时，应向玩家隐藏场景结构的细节内容。
- ☑ 当相机迅速移至某一新位置时，应保持相机与玩家之间的相对位置不变（第三人称相机）。亦即，当对相机实施重定位时，应保持同一控制（以及视觉）参考帧。多数游戏对此并未加以足够的重视，当相机进行重定位时，角色的移动常会导致玩家处于受阻状态。另外，上述现象还会导致玩家角色人物的死亡，抑或跨越当前区域的边界进而激活数据加载操作或产生其他延迟现象。当方向变化较为迅速时，根据既定的游戏体验以及玩家移动要求，基于上述方式的相机重定位操作其出现频率通常较低。当相机被迫穿越几何体和/或相机暂时（如 2 秒）失去玩家视线时，则可实施相应的重定位操作——这也可视为一种自动保护机制。
- ☑ 当玩家角色处于运动状态时，应避免将视线直接聚焦于其自身之上。该情况多发生于第三人称相机中，且目光通常直视当前玩家角色人物。亦即，玩家角色在屏幕空间内的位置保持不变——这在开始阶段尚且正确且玩家角色人物亦显示无误，但在多数情况下，该方案并不值得称道。通常，可通过玩家运动时的前视方向对场景中角色的位置以及行为动作进行预测。然而，前视值的确定过程往往较为复杂，具体内容将在第 6 章加以讨论。
- ☑ 待相机实现快速运动后，系统应保留相应的控制参考帧。针对基于相机控制参考帧的第三人称相机，相机的快速移动常会使玩家失去方向感，并导致难以预料的运动结果。最终，玩家很可能处于死亡状态。轻者，玩家对此会感到迷惑不解；重者，该游戏将永远失去一位玩家的支持。如果控制变化出现于游戏关卡的开始阶段，这将会导致玩家角色重返至上一关卡中以及加载延迟现象，并会诱发另一控制参考帧发生变化。
- ☑ 避免使用基于复杂几何体的封闭空间（针对第三人称相机）。相机的定位需得到足够的空间支持，以使玩家角色人物与场景游戏元素之间保持可见性。狭小、封闭的空间需将相机移至场景外部方可满足上述要求。如果场景空间与玩家角色十分接近，则会造成取景困难以及观察视线的频繁丢失。场景创建过程应从设计的初始阶段即考虑相机的取景问题。

本章小结

本章强调了相机设计的重要性：相机设计可视为游戏设计过程中的基本组成部分。针对大型项目，相关素材不可或缺，进而可将游戏设计方案整合至实时应用程序中。另外，该步骤应在设计流程的前期即加以考察。本章还阐述了相机的表现类型及其在实时程序以及非交互式视频序列中的选择方案。如前所述，2D 表现风格应确保场景的显示画面，真实反映设计者的初衷并满足一定的美学要求。相机设计并非易事，该过程可视为游戏设计的主要组成部分，并在玩家满意度方面发挥着重要的作用。成功的设计方案需要时间的积累以及精心的规划，且源于开发人员执著的探索精神。

第 5 章　相机设计方案

前述章节曾集中探讨了相机的基本表现风格及其各种行为，相信读者对此已有了较为深刻的理解。下面将对一些较为常见的游戏类型以及相机设计方案加以讨论。再次强调，相关选择方案并非一成不变，这里，也鼓励读者在设计过程中针对各种相机方案加以多方尝试。

游戏类型的定义颇具个人主观色彩，具体阐述方案亦多种多样。因此，制定一个统一的方案对于相机行为类型的定义是十分必要的。当然，此处所定义的类型方案亦并非是最终答案。另外，[Salen04]、[Wolf02]以及[Crawford03]包含了大量的与当前方案相关的可用素材。除此之外，众多在线资源也提供了庞大的素材库，其中包括[Lionheart]。

基于游戏类型的相机设计方案

下面将逐一列举较为常见的分类方案，且它们分别对应于不同的相机设计方案。如果可向玩家提供适宜的游戏画面，则相关方案即可应用于各类游戏类型中。

第一人称射击/冒险类游戏

第一人称相机（简称 FPS）常用于模拟玩家在真实世界中的各类体验方式，因而可向玩家提供逼真的视觉画面。当然，基于此描述方式的游戏通常并不局限于战争场景，游戏《银河战士》便是一例（Metroid Prime，[Metroid02]），且该游戏称之为第一人称冒险类游戏（简称 FPA）。尽管这一类游戏可视为较为直观的相机实现类型之一，但典型的 FPS 游戏也可能对此表现得较为隐晦，即并不直接向玩家展示较为明显的分类特征——这也是相机自身的定位所决定的，亦即，相机系统所饰演的角色不应喧宾夺主。下面将对 FPS 相机设计方案进行逐一讨论。

相机的定位方案

鉴于相机定位代表了玩家在游戏场景中的位置，因此相机的移动通常与玩家同步进行。相机的定位可通过相对于玩家原点位置的一段预置偏移量加以定义，即相对于世界坐标轴上的一段垂直距离。在某些情况下，该位置还可通过地标以及其他与动画或模型数据相关的位置或方向标记加以确定。

相机位置似乎应对应于玩家角色人物的眼睛位置（根据玩家角色模型的外视图），但实际情况并非如此。首先，相机的垂直位置将影响到玩家对场景规模、移动速度的感知程度；其次，场景中还包含其他渲染元素，例如武器装备、玩家角色的肢体，进而可大大增加玩家的临场感。这一类渲染元素通常具备一定的尺寸（以及位置属性）且与真实的游戏视口尺寸实现良好的匹配（以防止玩家的视线被

遮挡)。因此，相对于场景的规模，渲染元素并未采用其真实尺寸。另外，通常可对这一类元素加以简单的定位并确保其可见。在实际操作过程中，当从眼部位置进行观察时，上述元素难以得见，除非对头部位置进行适当调整。

技术要点

当采用第一人称相机进行观察时，武器装备的真实位置与玩家位置之间的不一致性在多玩家游戏中将会产生问题。这里，当玩家武器装备的发射物与玩家之间保持正确的相对位置，且从相机外部进行观察时（如其他玩家），二者之间将存在较大的缝隙。

针对于该问题，可在发射位置处添加光影效果抑或类似的图形效果以弥补这一缺陷。另外，在发射的初始时刻可能还需要对发射对象的渲染位置加以调整（至少在外部视图中），进而在一定程度上缓解或消除上述问题。如果发射对象的运动速度较快，则该问题表现得并不明显（这取决于发射对象是否在首次渲染之前发生移动）。

另一种可能的方案是选择相机垂直方向上的某一位置并获取足够的视野（相对于地面），以在陡峭的环境下简化跳跃以及漫游等操作。除此之外，相机垂直方向上的倾斜角还可自动进行调整（如斜坡、陡峭地面以及跳跃时的下行运动），以显示最佳场景画面。

相机的方向选择方案

由于第一人称相机体现了玩家位置处的视角，因此，相机的方向通常按如下方式加以设置：其可见视野类似于人行走或直立时的场景画面，即基于角色朝向的水平直视方向。第一人称相机常需要玩家对场景环境进行不断的探索，并对视野之外敌方角色人物的攻击做出适当的反应。因此，玩家针对相机的方向具有一定的控制权，该技术也称作自由视角。玩家的前视方向定义了一个参考帧，且围绕该方向玩家可实现自由视角功能。当采用自由视角时，若玩家的移动行为受到限制，则相机的重定向范围将形成一个锥体，其中，顶点位于玩家角色的眼睛位置处。

当采用自由视角模式时，若玩家的移动行为未受任何限制，则可对水平方向和垂直方向上的控制操作分别进行考察。其中，相机的垂直定向操作依然会受到上述限制。例如，相机的水平定向将与玩家角色同步执行，因此，经自由视角操控后相机将转向玩家角色。与常见的限制角色的角旋转速度相比，相关限制条件将不复存在。又如，水平自由视角操控将根据玩家方向对相机方向加以调整，这使得角色的头部和/或躯干的转动独立于腿部。自然状态下，自由视角模式适用于多种场合，其中，玩家可通过该模式表现各种科幻场景画面。

需要注意的是，自由视角模式对于第一人称相机并无过多限制，第 7 章将对此加以详细的讨论。另外，在第一人称相机中，相机的定向操作还会受到其他方式的影响。例如，在某些环境下，相机可辅助玩家实现垂直方向上的自动重定向操作。当玩家沿坡道上行或下行时，这将辅助玩家对相机实施相应的升降操作。类似的例子还包括，在跳跃动作结束时，相机应可自动向下调节倾斜使着陆区域部分可见。正常状态下，方向变化应尽量保持平滑且不能影响正常自由视角模式的使用。

武器系统的定位

玩家角色在 FPS 游戏中的表现常受到武器装备的影响，这里，武器定位时的选择方案往往出于美学概念以及可见性考量。

外部察看时的差异

在多玩家游戏中，其他角色眼中玩家的表现状态与其在场景中的位置并不匹配，尤其是玩家角色

武器装备的开火位置。

第三人称转换

当游戏在角色外部视角与第一人称视点之间发生转换时，应确保该过程实现为无缝变化，较为常见的方法是使用跳接镜头，当在场景环境中移动相机时尤是如此（或许还要考虑到几何体的复杂度以及距离等问题）。

缩放效果

多数 FPS 游戏的武器装备均可模拟瞄准镜效果。

周围视觉

大多数 FPS 游戏有限的水平视域（FOV）范围使得玩家的视野相对闭塞，进而限制了对敌方人物以及其他对象的观察能力。这主要是出于性能考量并可适当地降低广角透视变换所导致的画面扭曲现象。随着宽屏显示器的不断普及，水平 FOV 的范围也需要随之增加。当然，性能问题依然不可忽视。

瞄准位置

当 FPS 游戏采用相对于相机的静态位置执行射击任务时，针对于玩家的武器装备，则有必要对场景中的某一位置进行跟踪，通常，该位置由瞄准线加以确定，在某些情况下，该位置也取决于脚本机制以及玩家模型。

移动问题

相信读者对晕船所导致的呕吐、恶心这一现象并不陌生，对于某些玩家而言，FPS 游戏同样会产生类似的不适感觉。

相机抖动

在 FPS 游戏中，可通过相机抖动强调游戏对象的爆炸效果以及其他震动效果，进而体现玩家在虚拟世界中的画面感（如武器的反冲作用）。通常情况下，该过程需要多种效果的支持，如音效以及第一人称相机的移动效果。

相机摆动

当玩家在场景环境中行进时，FPS 游戏常会根据玩家的位置向相机添加一些移动量，相关移动通常为预定义路径或垂直位移，且与玩家的移动速度成正比。而某些游戏则会降低或移除这一类运动效果，特别是玩家的移动无法正确地反映人物在其方向上的行进或跑动状态时。偶尔，该效果也用于表现玩家处于受伤状态。需要注意的是，当采用这一类移动方式时，相机不应穿越任何几何体。更为重要的是，上述效果不应使玩家产生任何不适感。

空闲移动

类似于相机的摆动，某些 FPS 游戏在一段时间内将采用随机或半随机的方式以表示玩家处于休息状态，抑或某一阶段内玩家未接收任何输入信息。该操作通常会伴随着玩家角色武器装备的动画效果，以表明相关角色人物正在等待玩家执行后续动作。

调整运动比例

运动比例问题在 FPS 游戏中十分常见，即使角色人物的真实速度较大，但其在场景中行进时，

玩家并不会感到任何不适。例如，在游戏 Metroid Prime（[Metroid02]）中，主角的移动速度可达到 12 米/秒——该速度可轻易地打破百米世界纪录。在 FPS 游戏中，相机的位置大约在主角的眼部位置处，然而实际情况则是，该位置通常位于“正确的”物理位置上方某处，进而使观察者获取较好的场景画面。

技术要点

由于相机位置作为相对于玩家角色的固定位置加以计算，因而相机易受到各种影响玩家角色移动的各类因素的干扰，其中包括由输入设备或游戏场景元素所导致的任何轻微、短暂的位移，如表面突起或崎岖的对象表面。当角色在地形表面上移动时（无论采用第一人称相机或第三人称相机），玩家通常可直接穿越小型障碍物，此时，玩家并不会因这一类无关紧要的因素而停止。如果位移量较小，这可以通过在碰撞点上方位置处进行重定位操作加以实现。需要说明的是，允许的垂直移动量常称作提升阈值。玩家的垂直移动应体现出某种瞬时性，该过程可能仅维持若干毫秒，随后即恢复至其原始位置处。考虑到相机视野通常与玩家的位置紧密相关，因此，这将导致相机画面亦会在垂直方向上移动少量距离并会随之产生不规则的画面故障。

然而，该问题可通过在相机（而非玩家角色）垂直方向上添加一些阻尼效果加以解决。特别地，可适当地限制相机在垂直方向上的移动，并保持其平滑的运动状态且与玩家的移动方式无关。另外，阻尼效果仅适用于角色跨越游戏场景中的崎岖表面时。同时，通过在相机垂直运动方向上添加少许延迟，该效果还可用于增强角色的跳跃感以及着陆感。第 10 章将对阻尼效果进行详细的讨论。

需要注意的是，当玩家角色垂直方向上的快速移动需要第一人称相机与玩家之间保持固定时（如乘坐电梯时），则应避免使用阻尼效果。除此之外，第一人称相机中的阻尼值应小于第三人称相机。这里，应确保玩家对其附近处的相机系统实施正确的操作。相应地，第三人称相机通常距玩家角色稍远，因而可保证垂直方向上的延迟不会影响到玩家对于场景角色位置的理解。

角色类/动作类冒险游戏

角色类游戏展示了多种相机处理方案，其中，玩家以动画人物或类似的角色出现于游戏场景中（包括动作类冒险游戏）。在大多数情况下，可根据玩家的行为能力确定相应的相机设计方案。角色人物在场景中行进时所采用的方法将极大地影响相机的位置和定向操作。广义而言，玩家角色的行为能力可分为两种，即地面类角色以及飞行类角色。而某些交通工具既包含地面类角色也包含飞行类角色，因而其处理方案也稍显不同，具体内容将在本章后续内容中加以介绍。

地面类角色

针对于地面类角色，玩家角色通常在环境表面上行进，可能还会包含短暂的跳跃能力。另外，当玩家角色被场景中的突起物绊倒时，还应具备相应的跌倒能力。这里，典型的行进处理方案是在玩家角色后上方对相机进行定位。

当玩家角色转动身体时，相机可围绕玩家在水平弧线上加以旋转，进而维持二者之间的位置关系。在某些游戏中，相机在位置间执行关联操作时，貌似在相机与玩家角色之间设置了一个“连杆”，进而保证相机的定位与玩家角色的旋转之间可同步执行。但是，当相机具有较大的角速度时，上述操作定会使玩家失去方向感。这里，常见的解决方案是适当地限制相机围绕玩家角色旋转时的角速度，并在相机接近目标位置过程中（或角位移）向角运动添加一定的阻尼效果。另外，在确定相机位置过程中，适当地添加角度延迟可使显示画面更具美感和舒适感。首先，这将降低玩家角色快速旋转时所导致的方向感的迷失；其次，相机画面还将包含角色眼中的旁侧风景，以进一步强调玩家角色模型的三维

特征。

需要注意的是，上述延迟量不应过大，这将导致相机定位时改变控制参考帧，抑或阻止玩家与角色移动之间的可见性。针对于此，该问题可通过限制期望水平角度中的以及相机的最小、最大角位移得以解决。在第三人称游戏中，该现象屡见不鲜，如游戏《古墓丽影》（[Tomb96]）。

当确定相机位置时，另外一种方案是忽略玩家角色的方向。此时，当前方向表示为相机与目标对象之间的方向，针对于此，距离值与高度值依然可满足相关要求。鉴于相机并不会围绕玩家角色旋转至期望位置，因而玩家角色相对于相机并无方向性的限制。当采用这一类“全向性”的相机定位机制时，控制参考帧通常会根据相机与玩家角色人物之间的位置关系加以确定（相机相关），而非角色方向自身（角色相关）。通过该方法，玩家角色可按圆周轨迹旋转或移动，且无须强迫相机在场景世界中快速移动以维持其在玩家角色后方的相关位置。较为常见的例子是 Ratchet and Clank 系列（[Ratchet02]）、Super Mario 64（[Mario96]）以及 Super Mario Sunshine（[Mario02]）。

此处，应仔细考察相机与角色垂直移动之间的相互作用。如果相机的垂直移动与玩家角色相关，则最终的移动结果将处于不稳定状态，当玩家角色跨越崎岖对象表面时尤是如此。

这里，相机垂直移动上的阻尼效果可降低相机移动过程中的不稳定状态，一如前面所谈到的第一人称相机。然而，垂直方向上的移动不可过分缓慢，这将使相机滞留于视锥体外部的时间过长，抑或相机会突然转向以保持角色可见。

通常情况下，相机位于角色头部上方位置处并与之保持一段距离，以保持足够的环境空间量可见。当然，一些游戏还会将相机定位于角色的下方某一位置处。

飞行类角色和游水类角色

飞行类角色与游水类角色具有诸多共同点且可视为一类角色，尽管后者的移动速度较慢。针对玩家角色而言，该运动类型之间的差异主要体现在自由度之上，且在场景中的移动方式将使得相机的位置难以确定。上述两种角色皆可采用全向相机处理方案，尽管相机通常位于角色的后方。当游水类角色迅速改变其运动方向时（包括直接移向相机自身），全向相机方案工作良好。根据当前环境，如果水域被包含于较小空间内，则静态相机可视为一类较好的处理方案。如前所述，飞行类角色通常比游水类角色具有更快的移动速度，因此，除了相机的移动速度较快之外，其重定向能力也略胜一筹。在上述两种环境下，定位以及旋转延迟可极大地增加角色的加速感。若飞行类角色可实现摆动或倾斜等功能，则相机应对此适当地采取某些定向措施。然而，二者之间的直接关联可能会导致玩家感到某种不适，如恶心、晕眩等现象，因而应谨慎使用。

隐秘动作类游戏

较为常见的第三人称游戏类型是隐秘动作类游戏或近战游戏。根据前述基于角色的相机类型，这一类游戏通常会采用相应的变本。例如，相机应可提供表现当前角色与敌方角色之间关系的场景画面。其他问题还包括武器装备的使用以及与第三人称视角相关的显示问题。如果游戏设计过程中需要使用到精确瞄准系统，则可考虑使用结合了第一人称以及第三人称相机元素的混合型方案，具体内容稍后讲解。

作为一类可选方案，可根据玩家行为自动调整视点的模态相机。此处，相机系统可对武器系统以及相关选项做出适当的反馈。正常状态下，相机可在玩家控制下提供相应的位置以及行为选择方案，Splinter Cell 系列游戏即采用了这一类相机表现风格。隐秘动作类游戏的常见特征是采用环境定位以及

相机操控。例如，当角色接近墙面外角时，相机将移入某一位置（通常距离玩家角色较远），进而使得该墙角区域的视野可见，且玩家无须处于暴露位置。通常，玩家可在一定程度上对相机的方向（以及位置）实施操控，因而，当角色处于掩护状态下时，则可对当前环境区域加以探察。有时，当玩家携带武器并行进至暴露区域时，相机应可提供精准的瞄准功能。第一人称相机还应提供相应的可选方案，进而使得角色在掩护状态下亦可对外部环境进行察看。相对于玩家角色，可在原位置基础上添加少许位移量以模拟角色倚着墙角时的情形（除此之外，还可添加少许的旋转量以强调移动时的真实感）。

隐秘动作类游戏常需要玩家穿越某些密闭空间，因此会涉及相机的定位问题，并以此令玩家获取较好的环境视野。实际上，在这一类密闭空间内，玩家常会占据较大的屏幕空间，从而使得游戏画面效果大打折扣。

这里，常见的处理方案是针对玩家角色采用透明度或 Alpha 混合效果，以使游戏场景呈现出相对清晰的状态。当然，该方案可能会使显示画面缺乏某种自然感，但在当前相机定位约束条件下，该方案依然存在可取之处。

在某些情况下，相机还可移至位于场景外部的某一有力位置，以确保玩家具有更为开阔的视野。需要注意的是，该表现风格仅出现于特定场合，应确保某些不适宜的场景元素被隐藏（如朝向背面的几何体）。此处，当在场景内、外部相机实施切换时，应提供相应的图像和/或音频效果以强调外部相机的定位方案。如前所述，这一类转换的最佳实施方式是跳接镜头（具体内容参见第 4 章）。实际上，游戏体验过程中跳接镜头所提供的一致性方案均可视为玩家的重定位操作的有效辅助方案。

另外一种常见的方法是通过路径或其他直接控制方案对相机的移动实施预定义操作，如隐式表面（具体内容可参考第 8 章）。同时，静态相机也是另一种相机类型且通常情况下工作良好，但相机间的切换及其跟踪操作通常难以实现，除非使用跳接镜头，其原因在于：定位变化将会影响到玩家的控制参考帧，相应地，这将使玩家失去方向感。

3D 平台

3D 平台游戏可视为第三人称相机的用武之地，该类型游戏包含玩家角色视觉画面以及场景画面。同时，这一类变现风格亦对相机系统提出了诸多要求，尤其是相机的漫游以及定位操作。当然，这里所描述的相关技术同样适用于其他类型的第三人称游戏，但相机的定位机制常取决于场景空间的大小。

3D 平台游戏存在两种主要的解决方案，每种方案均根据相机针对动态游戏体验的反馈方式加以选择。此处，这一类方案可简单地划分为两种形式，即全自由度和预置方案，其中，各种元素可组合使用进而创建相应的混合行为。

全自由度

针对于平台游戏，较为常见的方案是采用全自由度相机，该类相机可对玩家角色的行为实施动态响应。除此之外，如果不与游戏体验需求或其他游戏规范发生冲突，全自由度相机允许玩家根据其行为对相机的位置（以及方向）进行操控。这里，全自由度相机可使用与预置相机相同的技术，具体内容稍后加以解释。尽管如此，全自由度相机依然可呈现为动态方式并对玩家行为做出适当的回馈。

玩家对全自由度相机的操控方式可采用以下几种形式：

☑ 相机定位可围绕玩家角色加以确定，因而相机的运动方式将呈现为平行于场景水平面一道移

动弧线。

- ☑ 相机亦可围绕玩家旋转，针对相机位置与玩家角色之间所构成的屏幕，相机可沿该平面形成一道垂直运动弧线。
- ☑ 相机距玩家角色之间的距离可根据玩家期望动作或游戏体验需求加以调整，该变化常位于预置距离与相关高度值范围内。另外，预定义位置间的转换还可通过即时切换，抑或一段时间内的插值计算加以实现。有时，该距离还将根据玩家速度或者强调某种游戏体验元素而产生变化，如某种较为特殊的动画行进方式。

游戏设计者可结合使用上述各属性。需要注意的是，场景环境元素都不应被忽视，亦即，相机操控应确保相机对象始终位于场景几何体内部；另外，玩家不可将相机移至某一位置，进而影响当前的游戏体验（例如，玩家角色的遮挡、相机方向的垂直排列等）。类似地，玩家亦不可将相机移至某一位置，这样会使玩家角色完全被遮挡。除此之外，相机还可通过快速移动至某一位置并使玩家保持可见这一方式实现自身的调整（如在玩家输入终止时）。注意，仅当操作简便且可大大提升玩家的认知感时，方可对相机的方向加以调整，并通过将玩家置于视锥体之外这一方式对当前场景环境进行考察。如果玩家可独立于相关角色对其视见内容进行操控，则有必要提供一类显式的重定向操作以使其可返回至安全画面；抑或，一旦玩家输入操作终止，还可令相机实现自动出现操作。

预置方案

在该类型游戏中，其他常见的相机定位（以及定向）选择方案为预置方案。其中，位置数据常由样条路径、隐式表面静态路点（采用位置插值计算以及定向方案）以及类似的技术加以确定。相关方法将在第 8 章加以讨论。获奖游戏“战神（[God05]）”即采用了该相机类型；游戏 Ico（[Ico01]）则采用了预置相机方案。

顾名思义，预置相机针对相机的定位以及定向提供了完全控制方案，进而确保游戏体验的可视内容更具意义且更为清晰，因而该方案异常高效。当玩家移动受到严格控制时（3D 平台游戏大多如此），针对场景内的玩家定位，预置相机方案应加以精心设计。同时，相机碰撞以及其他显著问题暂可忽略，这一点亦尤为重要，其原因在于：相机的漫游功能相对复杂且占用了大量的处理器开销。针对于此，预置相机方案颇具吸引力。

然而，并非所有玩家交互过程均支持上述方案，这也是预置相机令人感到遗憾的地方。而且，场景环境的变化还可能会增加项目的开发周期；游戏状态的变化亦会导致移动路径的改变。又如，玩家是否应涉足于未知环境？此时，预置相机则无法通过较好的视觉画面对玩家加以显示。鉴于相机被限制于预置路径之上，因而上述情形依然存在调整余地。同时，有必要对此进行反复测试，以避免该情形出现于正常的游戏体验中。

另一项需要考察的内容是预置相机的美学效果，应确保相机的移动、定向以及定位操作符合一定的美学要求，这将生成平滑、可控的相机移动效果。然而，考虑到玩家视觉内容大抵相同，因而这将产生一种感觉，即玩家并未对游戏具有过多的控制力。该情形通常难以评判：相机移动的重复性将削弱游戏的交互性特征——这可通过使用移动空间而非真实路径加以解决，抑或对相机的加速数据以及位置数据进行适当的调整。在上述示例中，相机的预置属性将被削弱（这也使得游戏画面更加生动自然），但操控性依然存在。

角色饰演类游戏/在线多人游戏

角色饰演类游戏（RPG）与多玩家在线游戏（MMOG）拥有诸多共性，且针对玩家以及场景环境均采用第三人称视野（例如游戏 Final Fantasy XI、World of Warcraft 等）。这一类游戏的相机系统通常可视为前述角色相机的子集。在某些情况下，该类相机还可使用预置路径或拥有固定的位置（以及方向）——由于视点位置保持不变且背景元素需要实现预渲染操作，因而后者在降低处理器开销方面表现卓著。类似地，鉴于图形硬件的限制，早期采用 2D 风格的 RPG 游戏均使用背景的“预渲染”方案。通常，背景图案由小型视觉元素或贴图元素构成（具体内容可参考“滚屏机制”一节），这一类技术多应用于正交投影相机中，详细内容可参考第 2 章。MMOG 游戏有时会使用第一人称相机并可提供较强的临场感，但是，这将使得角色间的交互作用变得更为困难。除了第一人称视角之外，第三人称相机通常亦作为一类可选方案以供玩家使用。针对于当前游戏体验需求，玩家有权利选择场景视觉画面的尺寸。

与基于角色的游戏相比，RPG 游戏以及 MMOG 游戏则具有较高的取景要求，针对于多玩家—游戏控制的角色而言，这常需要视觉画面得以同步执行。对象的取景操作是一类较大的话题且常需要与某些影视元素结合使用，此处，相机的定向方案可能需要相机调整其位置以确保游戏元素可见。同时，设计人员还应提供适当的场景环境以使相机可远离游戏对象进而完成所需的取景工作。因此，相机对象常置于游戏场景外部区域，而正投影相机尤是如此。

某些 RPG 游戏则赋予了玩家对相机定位和定向的完全控制权，并提供了相机的重定位选择权。同时，相机相对于主要（或活动）角色加以定位。针对于多玩家游戏，相机还应具备多玩家可视功能。

滚屏机制

针对于游戏场景世界，滚屏游戏常采用正交投影加以显示且与相对于玩家角色的观察位置无关，尽管类似的风格也可采用轴测投影加以实现。需要说明的是，该表现风格源于早期游戏所面临的硬件限制条件，相关设备通常不具备生成实时三维图像的能力，另外，其处理器性能以及内存空间亦十分有限。早期的游戏机游戏以及街机游戏甚至不支持位映射机制，因而仅采用贴图图像对游戏场景加以描述，即使用较小的、尺寸规则的图像加以显示并按照二维网格的方式加以排列，如启发类游戏 Defender（[Defender80]）。

针对于贴图硬件，多层贴图可彼此叠加（并辅以简单的透明度效果）。根据玩家的移动行为，各图层可以不同的速率加以移动，进而生成一定程度上的深度感。

随着游戏场景画面在贴图间的水平移动，背景图像则沿相反方向实现“滚屏”移动，亦即所谓的横向滚屏（卷轴）游戏。通常情况下，相机的位置（实际上是世界坐标空间内的视口位置）多由玩家角色的位置加以确定，该关系可根据移动方向、移动速度以及其他玩家角色的属性进行适当的调整。有时，相机将被禁止移至当前贴图范围的外部区域，抑或仅可显示部分贴图内容。另外，滚屏游戏常显示为某一方向上的持续运动，且分别包含单方向运动（例如右上至左下方向，又如街机游戏 Zaxxon [Zaxxon82]，以及游戏 Paperboy[Paperboy84]）以及随时间变化的多方向运动（如游戏 Spalt[Splat83]）。

上述持续运动迫使玩家角色停留于屏幕范围内（若移动速率大于滚屏速率），抑或玩家位置相对于

屏幕或滚屏方向保持不变。关于滚屏技术，读者可参考第 4 章“期望特征”一节以获取更多的内容。

除此之外，运动轴的不同也将导致视觉画面发生变化，如垂直滚屏。等轴测游戏可视为上述游戏类型的扩展风格，其中，背景图像亦由贴图元素构成，但相关图像则以伪等轴测方式加以绘制，进而模拟场景的三维效果。当然，其他伪投影方式也是可能的。这里，相机的移动通常锁定于场景世界的某一投影轴上，且相机将以适宜的像素步进尺寸沿该轴向进行移动。鉴于早期设备对资源的诸多限制条件，适宜的深度感可视为等轴测游戏流行的主要原因之一。关于等轴测技术以及其他投影类型，读者可参考第 4 章“交互式 2D 或 3D 相机系统”一节以获取更多的内容。

滚屏表现风格可扩展至包含某些三维视觉元素，无论是游戏对象抑或场景自身。相机的移动推出按照预置方式进行并被限制在平面内的某一路径之上。另外，相机的定位以及定向操作应谨慎处理，以在任何时刻均可生成游戏体验的最佳场景画面。当然，还可适当地添加特定的三维元素以进一步增强场景的深度感。

横向滚屏游戏中玩家角色的移动方式通常被限制在某一平面或路径之上，这一点与相机的操作方式类似。在某些情况下，背景元素还可视为一类三维凸起物，进而起到组织玩家运动的功效，或用于访问游戏场景中的不同区域。根据游戏体验的动态变化特征，上述凸起物亦可视为二维或三维场景元素，尽管其功能差异的表现方式较为困难。针对于此，相关行为的一致性不可或缺。

有时，游戏对象需要穿越（或置入）由背景元素构成的平面，因此有必要对该运动方式加以模拟。在二维游戏中，多个图层可通过类似于传统二维动画的方式对运动方向加以模拟（[Thomas95]针对传统动画技术进行了较为详细的讲解）。

竞技类游戏

竞技类游戏是一类较为风靡的游戏类型且涵盖了各类游戏场景以及游戏体验。多数竞技类游戏均具有结构清晰的封闭环境，因而更能集中体现相机所包含的各类问题。这一类游戏常需要对相关方案实施预计算，无论是室外游戏还是室内游戏皆是如此，尽管基于追踪功能的竞技类游戏与赛车类游戏拥有诸多共同点。

当表现竞技类游戏时，推荐采用电视风格的显示画面，其中包括介绍画面、重要比分以及相关报道的回放等内容。在某些较为知名的游戏中，通常会采用经过精心设计的相机方案来表现游戏场景。即便如此，其他辅助型相机技术依然不可或缺。例如，针对玩家交互操作，可通过最佳光照条件表现当前竞技类游戏。这里，相机在场景中的自由穿行大大提高了游戏的临场感。

相机属性常取决于当前竞技类游戏的真实感，亦即观众的临场感，尽管这对于游戏体验而言并非总是产生最佳效果。过分真实的表达方式需要相机参照真实设备的运动方式加以移动。非完美状态下的取景操作以及重定向操作可极大地增加现场氛围，并实现电视画面播放效果。此处，相机将对下列事实加以模拟，即摄影师难以跟上激烈的比赛节奏，因而导致最终的画面包含某些“缺陷”。

相机回放技术可对游戏事件序列进行复制并用于展示玩家的胜利（或失利）场景，因而该技术尤为重要。在相关事件的播放过程中，将采用与玩家角度完全不同的透视机制对游戏场景加以展示，这里，可预置或自动生成外部视角以展现特定的事件画面（如近景动作抑或围绕特定人物转动镜头），或者仅提供某一特定的拍摄角度。第 10 章将对上述自动过程加以详细的解释。一种较为常见的方法是将

相机置于场景内的某一固定位置，这一点在赛车类游戏中亦较为常见。其中，常采用正面看台这一角度来展示赛道上的车辆（以及多个与车辆相关的相机位置）。其中，特定相机的使用以及相机间的切换方式将根据设计者的意图以及（基于预置方案的）相机系统自动生成的结果予以实现（例如，相对于相机位置的目标对象的位置和距离）。

针对于各类竞技类游戏，是否存在一类通用方案用以实现相机的表现风格？答案是肯定的，该方案可将竞技类游戏划分为多个独立组加以实现，并对玩家的数量进行考察（单玩家、双玩家抑或团队作战）。

单玩家竞技类游戏

某些游戏在某一时间段内仅需要单玩家的操控和显示（如高尔夫游戏），这一类游戏简化了相机的需求条件，即使在玩家角色以较高速度集中球体时亦是如此。考虑到聚焦点通常为玩家角色或简单的游戏对象（如球体），因而单玩家游戏的取景要求往往较为简单。当球体被击中后，相机画面通常会跟踪球体的位置，抑或执行跳转镜头（或多次）以显示球体的最终抵达位置。

即使游戏中包含多位玩家，相机处理方案也可按照上述方式加以执行，当玩家未发生直接互动或轮流替换时尤是如此，如田径类游戏（跳跃动作、射箭动作等）。

双人或 4 人竞技类游戏

在竞技类游戏中，双人游戏屡见不鲜；某些游戏还会包含两组对手（如网球类游戏）。针对玩家而言，其他 3 位参与者往往由计算机加以控制——当然，这一类角色也可轻易地实现为玩家操控。这里，根据玩家的本地操作抑或网络连接特性，应对潜在的分屏显示问题加以考察，网络游戏分别将独立个体视为单玩家游戏，至少对相机而言应是如此。分屏渲染亦包含其自身的具体问题，具体内容可参见“分屏技术”一节。

团队竞技类游戏

在团队竞技类游戏中，玩家可在既定时间内对某一独立个体进行操控，也可在不同个体间进行相应的切换。通常，该过程可自动执行，以便玩家可对适宜的个体进行控制。例如，在足球游戏中，如果足球被踢至当前角色的前方区域时，控制操作可能会切换至距离足球最近的角色（或位置）。另外，控制权的改变往往还取决于具体的运动类型。除此之外，如无特殊情况，玩家应保持对当前角色的控制权。亦即，仅当对玩家明显可见时方可对控制操作加以调整。一类较为常见的方法是设置特定的控制命令以实现玩家角色之间的切换操作。无论如何，针对处于控制状态下的当前个体角色抑或可选控制角色，都应对其在屏幕中设置相应的标记，该标记需清晰显示且不可遮挡住相应的游戏元素。通常，这可通过改变个体角色渲染颜色加以实现。

团队竞技类游戏中的相机取景操作通常较为困难，当玩家企图与离屏角色发生关联时尤是如此。在某些情况下，玩家与目标个体之间无须可见，屏幕信息已然可传达足够的位置数据并辅助玩家实施战术决策，当消息传递机制较为完善时尤是如此。在某种程度上，当设计者在仿真以及游戏之间寻求

较为平衡的解决方案时，上述功能往往体现了其设计思想。

当不考虑个体角色的数量时，竞技类游戏可划分为两种类型，即室内游戏（场地类游戏）以及室外游戏。

室内场地类游戏

考虑到场地限制条件，室内竞技类游戏的场景描述相对简单，其主要原因是，相机的位置（以及方向）全部采用预置方案，进而可显示大部分场景的最佳画面。这里，设计者需要对场景几何体进行精心打造，其中，相机对象有可能置于当前场景的外部空间内。室内竞技类游戏应充分展现玩家角色的状态，而非处理几何体间的碰撞问题。针对于此，适当地削弱几何体的地位并不为过。

对于室内竞技类游戏，可通过真实的电视画面风格对游戏元素加以展示，这也是玩家熟悉、期望的一类表现风格。另外，以易于理解的方式表现当前场景元素不失为一种最佳方案。例如，玩家角色与其他参与者之间的关系应一目了然。当然，鉴于相关物理限制条件，游戏相机还可以一种异于真实相机设备的方式加以运动，但应尽量避免快速、混乱的移动以及定向方式所导致的失控感。

游戏事件的回放机制其设计方案应充分考虑到玩家的观赏习惯。再次强调，与电视转播相比，游戏相机的本质主要体现在其独特的运动方式（例如，球员得分后，相机采取环视方式拍摄）。另外，相机还可实现一类多区域同步显示模式，以向玩家提供某种辅助功能或作为一种特定的画面效果加以显示。

常见的室内竞技类游戏如下：

- ☑ 球类游戏（包括网球、篮球、壁球、美式壁球、排球、水球等）。球体的视线跟踪可视为球类游戏中一个较为重要的问题，电视转播画面常采用叠加图的方式描述球体的位置以及移动轨迹，该技术可方便地应用于游戏相机中，
- ☑ 桌面类游戏（包括美式台球、英式台球、斯诺克、乒乓球、撞球等）。桌面类游戏常需要提供多个玩家控制的游戏画面。当沿球体运动方向从视觉画面中执行移除操作时，球体的移动角度通常难以确定，这一点在第三人称相机中亦较为常见。当竞技空间较小时，则可显示全部桌面。针对某些桌面类游戏，考虑到其深度感通常难以评测，因而可向玩家提供辅助手段以简化相关控制操作（如球拍的近似位置或经适当简化的击球方案），抑或标出球体的运行轨迹。

室外竞技类游戏

室外竞技类游戏则包含了较为丰富的游戏元素，相应地，其相机处理方案也会随之发生。

（1）场地型游戏（如网球、美式足球、英式足球、橄榄球等）。这一类游戏通常局限于特定的场地中，这一点与室内竞技类游戏相近，当然，其场地规模远非室内游戏可比，类似的行为还包括针对于相机移动和方向的预置方案。然而，考虑到基于玩家操控的相机方案的多样性，因而场地规模应做适当调整，以使场景画面不致距离玩家过远，进而对当前场景加以充分的显示。

（2）全自由度游戏（如高尔夫球游戏、滑板游戏、滑雪游戏、水面竞技项目等）。类似于场地型竞技类游戏，这一类游戏通常也对游戏场地有所限制，但角色的移动以及定位方式更为多样化。相应地，其相机处理方案亦呈现为多元化状态。

（3）场道型游戏（如田径比赛、自行车比赛、滑雪比赛等）。考虑到相机完全可采取预置方案，因而这一类游戏的实现方式亦较为直观。这里，尽管仍可向玩家提供多个相机位置，但相机的场景漫游操作实现起来也并不困难。

抽象功能

并非全部游戏以真实状态加以展现，在某些情况下，游戏将对真实体育运动项目实施相应的夸大操作。例如，玩家的跳跃高度、击球力度都存在不同程度的增加。另外，游戏场景还可能位于独特的环境中，例如水下环境、外太空环境，相关操作与正常状态相比存在着较大的差异。另外，场景世界的物理特征也会随之发生改变（如重力条件、摩擦力等）。除非玩家采取特殊方式观察当前场景世界，否则，这一类变化并不会对相机方案产生过多的影响。例如，当重力为 0 时，游戏相机所产生的玩家画面将围绕其前向轴进行旋转。在大多数情况下，灵感往往来源于真实的比赛画面，这也使得竞技类游戏更具临场感。

赛车类游戏

尽管赛车类游戏涵盖了诸多竞赛元素类型，但仍然具备通用的相机以及游戏体验功能。通常，玩家被限制在赛道之上，进而形成相应的限制区域并包含起始点以及针对玩家移动的限制条件。自然状态下，上述过程可限制为赛道表面上的二维或三维运动方式，如较为典型的汽车比赛、飞行比赛以及帆船比赛。针对于场地型游戏，相机的位置以及飞行也可采用预置方案。然而在赛车游戏中，相机的位置多少会出现变化，并很可能通过样条曲线作为相机的移动路径，这一类运动方式稍显不自然并缺乏相应的人为操控感。当然，存在相关的方法可对相机的重定位操作进行调整（如采用第 7 章所描述的 PID 控制器），进而可有效地改善上述情况。

相机的回放机制可视为赛车游戏的一项重要功能。回放相机提供了前述游戏操作的非交互式画面场景。某些情况下，玩家允许在画面回放过程中针对相机的定位执行少许控制操作（甚至是全部相机位置以及方向上的操控），这可提供一系列的预置相机位置数据和/或方向数据加以实现。对于某些较为特殊的赛车型游戏，建议相机的表现风格应接近于电视转播画面。

相对于玩家角色，赛车类游戏通常需要向玩家提供其他赛车的相关信息，当竞争对手超越玩家且处于前方不可见状态时更是如此。显然，如果相机位于玩家角色后方过远处，则玩家会将相机对象误认为是玩家自身。然而，该相机位置将使得玩家车辆显得过于渺小，进而影响玩家的观察效果。

这里，一类可选方案是向玩家提供辅助画面，例如后视镜或画中画效果。另外，一种性能相对友好的版本则是动态更新全景地图，以显示全部赛车之间的相对位置。根据当前平面空间的尺寸，该地图可采用二维或三维图像进行渲染。其他视觉元素还包括相关警示图像，用于提示赛车间的车距，这一类图标元素通常于屏幕上方加以显示。

在赛车类游戏中，玩家可选择关闭或限定信息数据的显示数量，并确保玩家的注意力集中于游戏中。某些游戏仅提供真实车辆中的数据信息，如仪表盘。

赛车类游戏需要在真实感和易操控性之间做出选择，根据控制选择方案以及场景的渲染方案，该决定将影响游戏的方方面面，包括移动行为以及车辆的特点。在大多数情况下，非真实方案可提供较

大的灵活性，当玩家高速驾驶机动性能良好的车辆且缺乏实际经验以及相关技能时尤是如此。但不可否认的是，真实感依然在游戏中饰演了重要的角色，针对于游戏设计实现完美匹配的各类元素，应引起设计者的格外重视。某些特殊的赛车游戏需经过仔细的审核，毕竟玩家对此类游戏太过熟悉。针对于赛车游戏，Burnout 3（[Burnout04]）采用了一类较为著名的相机处理方案，该方案在实时游戏中添加了电影视频元素。

技术要点

视域可带来较为强烈的速度感并出现于大量游戏中。通过增加水平视域范围，游戏对象貌似以较快的速度与玩家擦身而过。由于该方案可导致画面的严重变形，因而应对其谨慎使用。另外，可适当地减缓视域的变化速度，以保证玩家获取正确的方向感。这里，较为常见的选择方案是分别对游戏对象和渲染显示画面使用运动模糊操作——针对前者而言，可采用累计缓冲区或其他多路渲染方案（参见[Akenine-Moller02]以及[Foley90]）；针对后者，则可在帧缓冲区中执行后处理操作并于最后进行玩家机动器械的渲染操作。最终，玩家机动设备将处于“聚焦”状态（亦即，假设玩家机动设备处于距相机镜头正确距离处且处于完美聚焦状态）。此时，背景元素则根据相机与玩家角色之间的相对速度呈现出不同的模糊效果。同时，玩家角色速度的不断增加还将导致背景元素越发模糊，通过这种方式，景深效果也可突出场景的移动感。

地面（或表面）机动设备

这一类游戏类似于角色类游戏，其中，机动设备所关联的相机通常置于其后上方，相机的移动在位置和方向上多通过延迟手段提供更为自然的场景画面，这也使得机动设备的方向可根据相机加以调整，进而强调机动设备的三维特征。同时，位置延迟是指车辆的位置可能会在屏幕中发生变化，因此应确保机动设备至少部分位于屏幕中，具体内容可参考第 8 章和第 10 章。

除此之外，某些游戏可能会在近车辆处提供相机视口，相关位置和方向使得玩家可快速判断其他车辆与游戏对象之间的相对位置。多数机车类游戏均提供了相机的旋转功能，针对基于角色的操控方案，这将使控制操作变得较为困难（如某些使用地面车辆的游戏）。另外，该类游戏常使得游戏画面定位于驾驶座舱或类似的结构中。实际上，这可视为传统的第一人称相机并包含全视角操控能力。

如果机动设备的方向具备动态变化、特技飞行以及自由翻转等能力，则在第三人称相机中应提供特定的处理手段，进而防止相机运动出现过于剧烈的变化。针对于此，可通过车辆的移动方向标识相机的位置，而非车辆自身的方向。当机动设备失控时，还应添加相应的阻尼效果。

即时战略游戏

即时战略（RTS）游戏提供了多种场景画面的察看方式，例如小规模游戏对象画面、包含多个游戏对象的中规模战术场景画面以及大规模的战略场景画面（可能涉及全部游戏场景）。通常情况下，RTS 游戏采用固定的相机方向以及预渲染的图像数据。出于清晰度以及性能考量，当采用等轴测相机时尤是如此（类似于前述滚屏游戏）。随着 GPU 功能的不断强大，2D 风格的即时战略游戏已变得势微，但在某些情况下，考虑到真实感问题，当车辆以及其他游戏单位跨越场景并采用三维方式进行渲染时，背景图案依然会使用到预渲染图像。

当采用全自由度相机时，可在预置视觉画面以及玩家操控视觉画面之间提供某一界面以供玩家选择。

飞行模拟

飞行模拟可视为车辆模拟器的一个特殊子集并包含自身的相机方案以及操控方案。尽管该类型游戏要求玩家应熟悉飞行操作以及驾驶舱环境，但相机方案不应仅提供与此相关的画面。在某些情况下，外部相机画面亦不可或缺且这一类场景画面通常需要一定程度的玩家操控的支持。

针对于飞行模拟器，相机是否具备旋转功能以匹配飞行器的飞行可视为一项较为重要的操作。对于飞行器驾驶舱内的视点，相机应匹配于飞行器的当前方向。然而，当采用外部相机画面时（或在飞行器局部坐标空间中的某一特定偏移位置处），应提供相应的选项以保持与世界坐标系相同的 Up 轴。需要注意的是，大多数玩家并不熟悉真实驾驶过程中的复杂性，并对场景中位置和方向感缺乏应有的体验；另外，当操控该类型的视频游戏时，多数玩家均会产生晕眩、呕吐等不适应感。通常，这也可视为显示媒介的一个功能项（即投影至二维平面上的、快速移动的三维场景世界）。

当飞行器穿越虚拟场景时，常需要相关视觉元素以引导玩家游历当前场景环境，以及相应的参考平面以辅助玩家对飞行器实施定向操作。当缺乏较为明显的参照物时，常会导致玩家失去方向感。这里，人工水平仪可在一定程度上起到相应的辅助作用，并根据具体环境呈现出不同的形态。虽然某些基于空间的游戏并非是真正的飞行类游戏，但同样展示了相同的游戏元素且对应的视觉元素亦大同小异。实际上，考虑到这一类游戏缺乏较为明显的方向特征（如上或下），因而上述视觉属性以及元素则显得更为重要。

这里，需要对玩家飞行器的控制操作实施进一步的考察。对于“真实”的飞行模拟器而言，控制操作通常较为复杂且手续繁多，并在很大程度上反映了真实设备的操控方案，即使最终输入机制稍显不同。另外，当对上述操作进行简化时，依然需要考察飞行模型的真实性。随着处理器性能的不断提升，仿真过程亦趋于复杂化，但在目前商业游戏市场中，飞行模拟器尚无法对全部特征实施有效的建模，如机身处的气流。

真实的仿真过程并不等同于游戏设计。由于大多数玩家的飞行经验主要源于相关影视作品，抑或作为一名乘客时的飞行体验，因而在大多数情况下，玩家所期望的游戏体验与真实飞行过程并不吻合。针对于此，游戏飞行器的行为与汽车更为接近，尤其是转向操作。与此同时，这也产生了一种基于平面的观察方式。大多数玩家并不能很好地处理处于旋转状态下的相机，相机常呈现为颠簸状态并最终下沉。另外，参考帧的不一致性（如水平仪）也常导致玩家失去相应的方向感。

在某些时候，维护相机与场景 Up 轴之间的一致性关系可获得较好的操控效果，亦即，玩家可正确地确定二者之间的上、下方向关系。除了相应的回放技术之外，真正的飞行模拟器通常不会采用上述处理方案。

具有真实效果的飞行模型

针对真实物理受力状态下的飞行器行为，具有真实效果的游戏飞行模型通常能够生成精准（或相对准确）的模拟效果。这一类游戏一般通过座舱视角加以观察，其效果类似于影视作品中的空战画面。同时，游戏还应在座舱内提供多个观察视点（如肩部上方位置、后置位置、侧面位置等），相关内容还包括不同距离、高度以及角度处的外部可视方案。外部相机方案亦建立在玩家操控基础上，其中，相

机将围绕飞行器移动，该方式类似于球体表面上的运动。如果飞行器具备翻转飞行的能力，则还应进一步维护相机与飞行器之间的相对位置和方向关系，以使场景呈现为颠倒画面。

非真实相机模型

非常规状态下的运动方向/自由度则是非真实相机模型的用武之地，如太空舱、飞行生物等。上述真实相机模型中的大部分内容可应用于此，此处，难点在于飞行器相对于玩家的逆向飞行，例如倒置飞行或平行于场景 Up 轴的垂直飞行。另外，飞行器或生物的物理状态可能并不符合真实的飞行模型，因而需要特定的相机移动方式以避免玩家失去方向感。

人工水平仪

大多数玩家可能并不熟悉飞行器的具体操作流程，甚至难以适应其定向方式（如仰角和倾斜角），针对于此，驾驶座舱内通常配有相应的仪表装置（如陀螺仪），有时也称作人工水平仪或状态指示器，可辅助飞行员对仰角以及倾斜角进行调整。类似的设备也出现于飞行模拟器中，其他飞行模型还可能使用 HUD 或屏幕图像显示同一类数据信息。

运动感

多数飞行类游戏均会涉及高空飞行状态，这将导致视觉与真实飞行速度之间的差异。当不存在相应的参考帧用以测算移动行为时，玩家常会感到飞行器的移动速度十分缓慢。再次强调，过度的真实感并不一定会提升玩家的游戏体验。这里，飞行器的运动感可通过多种方式加以改善，具体的选取方案可根据游戏的类型以及设置条件加以确定。通常，粒子效果可用于强调某种运动感，如云彩或飞行尾迹。这一类效果可作用于玩家的飞行器之上，或作为一类独立的模型并结合透明度加以使用，进而可标识相关运动方向。太空类飞行游戏常采用漂浮的残骸（独立模型或粒子/广告版效果）以提供相应的运动感。除此之外，音频效果也可用于强调速度的变化，例如引擎发动状态的变化、多普勒效应以及物体高速移动时的呼啸声。同时，视域效果也适用于飞行模拟游戏中，具体内容可参考第 2 章中的相关内容。

冒险类游戏

冒险类游戏以丰富的故事情节而著称，且随着技术的不断进步，其表现方式亦趋于多元化——从早期的文本型冒险类游戏到目前的全三维场景游戏，其中，较为常见的元素是玩家与游戏环境间的交互过程。早期的文本型游戏对玩家所输入的文本进行解析，或者向玩家提供相应的选项菜单，但玩家通常无法与显示图像实现二者之间的交互过程（除非玩家行为动作导致相关图像发生变化，如门的开启和关闭）。

在随后的冒险类游戏中，点击方式的出现可视为一类较为重大的变化（包括显示图像、鼠标或其他输入操控设备）。亦即，可提供操控鼠标或其他设备以实现玩家与游戏环境之间的交互过程。该类型

冒险游戏一般使用正交视图以反映游戏场景的视觉画面，当游戏对象进入或离开当前场景并对其进行缩放操作时，可适当地添加某些三维元素。除此之外，相机还可在游戏场景中移动（或摇动）并跟踪玩家的位置。

即使当前设备不支持硬件加速，某些冒险类游戏依然可使用预渲染图像以展示基于丰富细节内容的三维场景画面（其中，较为知名的游戏是 Myst[Myst93]），即使玩家无法对相机的方向实施有效的操控，足够的细节内容依然可构成相对完整的游戏场景。

随着处理器以及图形硬件加速设备的不断发展，3D 冒险类游戏大量涌现且多采用第三人称相机加以表现，尽管相机定位方式一般遵循于脚本相机移动以及定向方案。

通常情况下，冒险类游戏受故事情节驱动，且常通过特定的电影视频方案对相机的定位以及移动方式加以强调（当与游戏角色人物交互时更是如此）。鉴于冒险类游戏的节奏一般比较缓慢（其行为常通过谜语的方式加以展现），因而相机可利用这一特点向玩家适时地展现某些电影视频元素。

拼图类/Party 类/棋盘类游戏

拼图类游戏以及 Party 类游戏在表现风格上具有某些共性，在多玩家分屏显示方面尤是如此，棋盘类游戏则可转换为相应的视频游戏格式，类似于 Party 类游戏，该类游戏亦支持多种表现风格。拼图类游戏在游戏内容、玩家操控以及表现风格方面均呈现出多样性，例如，与玩家之间的交互频率从回合制游戏至实时游戏不等；玩家数量从单玩家至多玩家不一而定。从本质上讲，大多数拼图类游戏并不需要突出玩家特征，玩家仅需要对相关游戏对象进行适当的操控并破解谜语即可，例如十分经典的游戏俄罗斯方块。这一类游戏的共同特征之一便是采用二维方式加以表示，其中，全部游戏区域在单一屏幕范围内可见，即使对多玩家亦是如此。此处，二维方案涉及较少的相机内容，除非游戏区域大于可视屏幕区域。针对这一类情况，只需简单地移动相机即可。

然而，考虑到显示设备的某些限制条件，玩家很可能在某一时刻期望获取最佳观察点，因而包含 3D 元素的拼图类游戏允许玩家参与少量的控制操作。一类替代方案是操控对象自身而非相机，某些拼图类游戏则采用此方法简化玩家的控制操作。另外，根据游戏的特征，该过程可自动执行；抑或提供相关辅助措施以向玩家显示下一个感兴趣的游戏对象；或者在必要时自动执行对象的旋转操作。上述允许范围内的相机控制操作可针对相机位置采用预置方案，或者直接执行围绕关注对象的相机旋转操作。在某些情况下，有必要针对相机的位置以及方向提供全方位的控制操作。

某些拼图类游戏将相机操控作为该游戏的一部分内容，此时，相机需频繁移动以配合玩家对游戏元素的控制。另外，相机定向操作亦可作为交互过程中的重要组成部分。在某些游戏中，相机会在一段时间内指向某一对象，则玩家通过该方式标识该对象。

一些拼图类游戏将玩家与游戏元素分离开来，进而彰显该玩家的特征。此时，玩家角色直接穿行于场景中，进而引发相关元素的变化；或者，游戏场景元素处于操控状态下，而玩家则根据其变化产生相应的移动。该思想取材于真实的拼图类游戏，亦即，通过改变游戏元素的排列方式间接地致使玩家角色移动（如滑块拼图游戏）。

较为简单的拼图类游戏使用静态相机进行定位以及定向操作，其中，全部拼图可同时进行察看，如游戏俄罗斯方块。通常情况下，多玩家拼图游戏将采用该方案，以便各玩家可观察到彼此的游戏进程，针对使用单显示设备的多个玩家尤是如此。

在拼图类游戏中，相机的移动通常被限制在平行于场景坐标轴的某一平面之上；而在某些情况下，相机还可穿越该平面进行重定位。

格斗类游戏/近战游戏

格斗类游戏则是另一种较为流行的游戏类型，从早期的二维、基于精灵的街机游戏至包含丰富的图形渲染内容的三维计算机游戏中均可看到它的身影。这一类游戏通常仅包含两个角色，因而在一定程度上简化了角色间的交互过程。同时，这还将大大降低性能方面的需求并进一步丰富角色中的细节内容。另外，角色数量的限制使得某些玩家控制操作可以隐式的方式执行（亦即，角色间可共用同一命令，如攻击命令）。但在某些格斗类游戏中，也存在玩家在某一时刻难以确定攻击对象这一情况。

近战游戏通常在较为开阔的环境下进行，且角色与场景世界之间存在较少的交互（偶尔，墙壁或游戏对象被摧毁）。有时，场景中可能会包含某些危险区域，因而给玩家的场景漫游带来一些困难（无论是有意还是无意），严重时甚至会使玩家“性命不保”。除此之外，在正常的游戏体验过程中，某些游戏事件（如摧毁墙体）可能会触发非交互式视频序列的播放，针对这一类情形，也存在多种具体操作方式（如河面中某一浮动的平台），但原则上应保持场景环境的简单性，这将导致处理器的消耗大幅下降。目前，格斗类游戏无论是在物理外观还是动画制作方面，其场景描述愈发真实。当角色间发生碰撞时，逆向动力学以及对手间肢体、武器冲突等交互过程均会产生较大的处理器性能开销，因而场景的简化处理具有重要的实际意义。随着游戏平台的不断强大，相关研究亦不断深入，玩家角色与场景环境之间真实可信的交互过程则指日可待。

格斗类游戏常置于开阔场景的另一个原因是，该过程中的相机的定位更加直观，针对主角的取景操作尤是如此。采取适当的方式对角色实施动态取景操作常会涉及相机的移位操作，其中，对手角色间的距离越大，则相机的移动距离也就越远，尽管增加视域范围在一定程度上可起到环节作用。游戏相机的动态移动通常并不是期望的结果，因此，应对对手角色预留足够的空间并对相机进行统一定位，即使角色采用某种极端的移动方式，也应使其保持可见。需要注意的是，应保持游戏角色的尺寸足够大，以避免细节内容的丢失。

关于相机与角色间的定位，一种替代方案是采用分屏或多视口方案显示与敌方角色之间的相对位置，例如，可通过画中画这一类小型视口展示游戏的正面场景，且与当前游戏状态紧密相关并始终处于显示状态。当然，分屏技术可以一种较为清晰的方式展现当前游戏场景，即使屏幕空间尺寸被减至“非标准”状态。视口的另一种用途是显示格斗的近景放大效果，该效果可大大提升玩家的满意度。

格斗类游戏严重依赖于角色的表现风格，尤其是战役的领军人物抑或是获胜后的表现场景。针对于非交互式视频序列播放过程中的取景以及相机的移动，这里则是电影视频技术的用武之地。特别地，格斗类游戏常通过对相机的移动和定向进而强调各种移动模式以及游戏过程中的搏击动作。在多数情况下，相机的移动模式均经过精心设计，冗长且仅包含部分交互特征。交互性的缺失使得相机的定位操作可以最佳方式展现其移动效果，并使玩家的注意力完全集中于角色本身。鉴于在特定的移动过程中角色间的相对位置已知，因而相机的位置与方向可在其工作过程中保持较好的一致性。这里，假设相机在移动过程中占据足够的空间且不会跨越当前场景环境。

在格斗类游戏中，当强调角色的详细运动方式时，一类常见的技术是通过快速视频序列对角色的动作进行多次回放。当一方角色获胜时，该情形屡见不鲜。此处，较为简单的方案是在首次播放时对

相关序列实施精准的回放。此处，可对画面进行适当的调整以获取更加有趣且精美的场景内容。另外，简单的电影视频技术也可应用于相机的定位操作之上，并根据当前环境实施对应的动态变化方案。考虑到格斗类游戏多发生于开阔的环境中，因而该类游戏可与动态电影视频技术实现较好的结合。一类替代方案是（该方案实现起来也较为直观），根据角色的胜负结果确定相应的相机移动选择方案。在某些情况下，游戏角色可能会被挤出格斗舞台（也称作出界），此时，动态相机处理方案亦不可获取。当然，具体情况还要视当前场景环境特征而定。游戏 Soul Calibur（[Soul98]）提供了较为完美的格斗类相机解决方案。

多玩家相机处理方案

若不考虑游戏类型，当多个玩家共享一台显示设备时，多玩家游戏在显示游戏场景方面则显得捉襟见肘。相关问题主要源于有限的场景显示能力，此时，各个玩家执行相应的动作行为，且对其实施独立的定位操作并获取各自的游戏场景画面。相应地，网络多玩家游戏（或基于物理连接的游戏机设备）通常不会面临这一类问题，皆因各个玩家均会配备个人显示设备。针对多玩家游戏，某些游戏还会支持多显示设备。

当在单显示设备之上展示多玩家游戏时，针对相机设计者而言，存在多种方案可供其选择。在多数情况下，上述选择方案应遵循于具体的游戏类型，且不同游戏类型均包含相应的显示特质以控制对应的角色人物。然而，多数第三人称多玩家游戏均采用单屏显示方案。协作型第三人称游戏或 Party 类游戏多采用单屏显示方案，并结合当前场景环境以阻止角色超出显示范围。针对于单屏显示模式的限制条件，当前游戏如何向多玩家进行展示？一种方法是采用轮换政策，即在一定时间段内，显示设备供某一玩家占用，且各玩家轮流转换并处于活动状态。虽然该方案较少出现，但针对 Party 类游戏却运转良好，其中，玩家与回合制角色饰演类游戏之间的交互过程并不明显。早期的街机游戏多采用此方案，皆因各玩家游戏均视为完全独立的个体。

当多玩家与单显示设备实现同步交互时，仅存在两种可选方案：分屏显示模式，即将屏幕划分为多个区域（每名玩家占用一个区域）；抑或，仅显示游戏场景的单一画面，且包含全部玩家，读者可参考第 4 章以获取更多内容。某些早期的单屏多玩家游戏包括 Super Sprint（[Super86]）以及 Gauntlet（[Gauntlet85]）。

单玩家游戏通常得益于显示设备全画面的使用，进而向玩家展现游戏场景中丰富的细节内容，甚至可同步显示多个视觉画面。针对于游戏设计者以及细节设计者而言，基于单显示设备的多玩家游戏通常会面临诸多难题。针对这一类风格迥异的表现方式，读者可尝试回顾本章前述内容以获取不同的表现风格。

当在同一台物理设备上向多玩家展现场景世界时，多玩家游戏的设计过程通常较为复杂。若玩家拥有独立的显示设备，且考虑到第一人称游戏具有较为强烈的临场感，因而该类型游戏通常工作良好。然而，大多数第一人称游戏均在计算机设备或游戏机设备间提供了网络连接以支持多玩家用户，因而无须对相机系统进行调整。然而，在某些情况下，多个玩家需要在某单一设备上实现同步操作且无须考虑相机的表现风格。

当采用单一显示设备展现多玩家游戏时，分屏技术可视为一类较为常见的方案。由于玩家将占据

各自的显示区域并获取相应的游戏场景，因而可视作彼此独立的渲染画面。

在 2D 或 3D 游戏中，多玩家游戏可采取多种实现方案。其中，相对简单的一类方案是单玩家、单显示设备，且与单玩家游戏具有相同的格式。当多个玩家需在同一台显示设备上加以展示时，则情况将变得愈加有趣。针对于此，下列内容列举了几种可能的选择方案：

- ☑ 单视口。此处，所有玩家共用显示设备的同一区域，通常为全屏显示画面。如果游戏区域大于静态、定向相机所能显示的区域，该方案将在对玩家取景时产生问题。
- ☑ 静态多视口。每一位玩家均拥有各自的显示区域，该区域可通过全部显示画面除以玩家数量得到——这是一类较为常见的处理方案，当采用第一人称相机风格时尤是如此。当然，屏幕空间尺寸毕竟有限，这将给玩家对场景的观察造成一定的困难。另外，遍历场景的层次结构并确定玩家视口中的可见几何体也将会消耗额外的处理器性能。
- ☑ 可变视口。在某些第三人称相机游戏中，当玩家彼此接近时，可采用单一视口；而玩家彼此分离时，则可调整为多个独立的视口。在实际操作过程中，该方案虽难以执行，但并非一无是处，针对协作类游戏尤是如此。首先，该方案有效地克服了玩家逆向移动时的取景问题；其次，当玩家彼此接近时，上述方案可获得较好的游戏场景画面。当然，该过程应确保视口转换以较为平滑的方式进行。同时，当玩家在边界处反复移动时，还应尽量避免视口间的碰撞。

在分屏多玩家游戏中，若玩家位置间逐渐远离，应存在足够的内存空间以装载同步显示的两个独立区域；同时，由于两个区域间基本不存在共享数据，因而该情形还会影响到数据与大容量存储设备之间的吞吐率。

单屏技术

针对于多玩家游戏，在某些场合下，全屏处理方案无法避免，甚至常作为首选方案。通常，鉴于各玩家需要对角色进行同步控制，因而该方案仅适用于第三人称相机。当确定玩家处于活动状态时，回合制游戏通常更适合采用全屏方案。

格斗类游戏一般通过单一画面显示当前游戏场景，而不考虑角色的数量。当选用适当的相机定位方案并以清晰、适宜的方式对角色取景时，这将导致问题的复杂化。协作类游戏一般也会使用全屏方案，以确保所有玩家获取相对清晰的场景画面。同时，该类游戏常会限制玩家的移动，以使其位于场景的可见范围内，进而可解决大多数角色取景问题。但此处存在一类特殊情况：根据当前环境结构特征，两个玩家可能会行进至某一位置且导致二者皆无法移动。如果玩家皆处于离屏状态，则有可能出现这一现象。该情形也可视为一类安全测试条件（最终，某一玩家可以异步方式移动至相应的“安全”位置）。防止该情形发生的有效方法是对当前场景环境进行适当的调整。

针对多玩家全屏游戏，下列各项显示了需进一步考察的内容：

- ☑ 相机的定位难以确定。针对于此，一类常见的方法是将所有玩家的中心位置作为参考帧，进而实现相机的定位操作。该方案适用于非透视投影，即使相机架于较高位置处且获得全部玩家画面时亦是如此。
- ☑ 非透视投影相机（如等轴测投影）。由于等轴测相机的方向保持不变，因而其世界空间坐标可根据玩家角色的屏幕空间位置获得。随着玩家角色的移动，相机无须执行重定位操作，第 6

章将对运动延迟问题进行详细的讨论。通常，可对相机的移动进行某种限制，进而避免不必要的抖动现象。针对于当前场景世界，等轴测游戏通常在某一固定高度处架设相机且与玩家的移动方式无关（如跳跃动作）。

- ☑ 相机的快速定位可导致方向上的迅速变化。该情形通常可以避免并令相机在距玩家角色较远处进行定位。大多数格斗类游戏将根据玩家的动作调整其位置或视域范围。例如，当某一玩家遭受攻击后暂时处于失效状态时，系统通常会采用相应的视频方案用以强调玩家遭受重创时的状态。如果相关视屏序列具有非交互式特征，这一类相机变化还是允许的。然而，当玩家数量超过两个时，该技术则显得捉襟见肘。

分屏技术

当采用 2D 表现风格时，多玩家游戏通常表现平平，这一类游戏体现了玩家之间的协作性以及竞争性。如果无须移动相机即可显示全部游戏区域，则背景可采用预渲染方案并可视为不变元素，这将大大降低 CPU 或 GPU 的性能开销。当图形硬件支持图层间的合成操作时（如多重纹理贴图），则无须对背景数据进行更新操作。

若游戏区域大于显示范围，则应采取相应方案以确保全部玩家可见。这里，可沿垂直于游戏区域的某一轴向上移动相机即可，该效果等同于“缩小”操作。另外，上述操作还可在游戏过程中实施动态处理，尽管这很大程度上取决于目标平台上的硬件渲染能力。另外，如果角色尺寸使其在当前场景环境中具有足够的自由度，则场景可采用固定的屏幕尺寸加以渲染。当采用固定的渲染尺寸时，基于当前场景的屏幕窗口位置应顾及全部玩家的视野。如前所述，这将导致屏幕产生持续的运动，这并不是我们期望的结果。更加符合美学要求的对应方案是适当地阻止相机的移动行为，与投影至场景世界中的屏幕空间的物理边界相比，直至玩家移至阈值距离范围内。在 2D 相机系统中，当采用正交投影时，该计算过程相对简单，即使视口围绕场景坐标轴旋转亦是如此。

如果通过上述方式对角色取景，可能会出现一个或多个玩家滞留于某一位置且最终将被移出屏幕范围。针对于多玩家游戏而言，则有必要考察每一位玩家是否均拥有其自身的视口；抑或是否拥有相同的全屏视口。针对于前者，可分别渲染各自的视口且该过程相对直观。

针对于分配方案，下列内容列举了相关注意事项：

- ☑ 针对于每一位玩家，屏幕划分方案应提供均等的屏幕空间，通常，可在视口间进行水平或垂直方向上的划分操作。在某些情况下，可向某一特殊玩家提供较大空间的视觉画面，特别是在协作类游戏中（例如，某一玩家负责操纵飞行器，而其他玩家则负责控制炮塔）。
- ☑ 玩家的数量也间接地表明了屏幕的划分方案。如果某一玩家在当前游戏进程中突然加入，则屏幕应根据目前最大玩家数量进行划分。当然，也有可能根据当前处于活动状态的玩家数量而适当地增加视口的数量。
- ☑ 在某些游戏中，场景世界可能过于宏大，因而对应数据不会一直位于内存中。通常，当玩家遍历场景世界时，游戏会从外部存储设备中读取数据（多数时为 CD 或 DVD，在某些情况下则通过硬盘甚至是网络读取）。针对大型环境，该方式可提供更为流畅的场景体验。在多玩家游戏中，各玩家可能会根据不同的方向遍历当前游戏场景，因而难以从外部存储中快速获取相关数据。在单玩家游戏中，相关数据很可能设置于外部设备中，进而对流读取过程加以优

化；而对多玩家而言，外部设备会重复搜寻不同位置，因而在数据加载过程中，游戏将会产生延迟现象，直至数据加载完毕。一种可能的解决方案是对关卡设计做出某种限制，以使各玩家仅遍历游戏场景的同一区域（该范围应足够大）。同时，这也要求玩家以同一方向遍历当前场景世界，或许，玩家之间还需要彼此协作以共同搜索关卡的出口，期间不应遗落任何一名玩家。

- ☑ 当采用分屏方案时，则有必要适当地减少视觉画面上的额外信息，原因如下：首先，各个玩家的视野其尺寸均有所降低，因而应减少或简化屏幕上的信息；其次，屏幕空间的减少意味着玩家画面屏幕分辨率的降低，这将增加玩家对细节内容的分辨难度。
- ☑ 当对较为重要的玩家信息位置进行设计时，应避免将相关信息置于显示设备的边缘处。这里，边框区域通常称作安全区，而边框内部区域则称作安全框，具体内容可参考第 2 章中的相关内容。另外，还可适当地减少渲染视口的高度，继而获得性能上的提升。

根据屏幕空间的划分方案，很可能出现下列情况，即玩家视觉画面与显示设备相比具有不同的宽高比，因而应对视锥体、视域以及其他与玩家视野相关的渲染条件进行适当的调整，以避免场景画面产生变形。另外，广角视域很可能会导致鱼眼变形效果，当然，这并不是期望中的结果。

全屏与分屏之间的转换

根据游戏体验需求，很可能需要在分屏技术与单屏技术之间实施相应的切换操作。如前所述，若多玩家可以任意方向在场景中移动，这将导致问题的产生，因而将游戏显示画面划分为多个部分可视为一类较为有效的处理方法。

下面首先考察分屏与屏幕合并的操作时机。这里，在连续执行分屏/合并操作之前，应确保存在相应的时间缓冲或物理空间。

- ☑ 超出角色间的距离（一个或多个角色）应保留对应的可见屏幕区域。
- ☑ 将屏幕划分为两个区域（可根据需要执行水平方向或垂直方向上的划分操作）。此处，在划分过程中，应确保玩家角色保持相同的屏幕相对位置，因而在切换过程中需要对相机的位置进行仔细的编排。待分屏操作执行完毕后，玩家角色的屏幕位置则相应地发生变化。根据游戏体验需求，该切换操作可对玩家的新视觉画面进行重定位。
- ☑ 在切换过程中，用以显示玩家能力值或分数的相关视觉元素其位置都应进行适当的调整，以适应于最终的视口尺寸。
- ☑ 随后，可对视域进行插值计算，进而与玩家的当前视野匹配。
- ☑ 必要时，还需要对相机的位置进行插值计算，以获取基于当前视野的最佳位置。

分屏与全屏之间的转换

一旦玩家角色彼此靠近，画面应切换至全屏显示模式。理想情况下，该切换操作发生于角色接近真实位置时，这将在一定程度上消除玩家的混乱感。

- ☑ 当玩家足够接近时，即启动切换操作。
- ☑ 可移除 GUI 以及其他视觉元素，直至切换操作完毕。

- ☑ 新相机位置应确保各角色保持当前屏幕位置。
- ☑ 针对全屏显示，需执行相应的视域插值计算并获取新值。
- ☑ 针对相机位置执行插值计算并获取由玩家角色确定的期望位置处。

本章小结

本章讨论了常见的游戏类型以及对应的相机解决方案，相关方案颇具实用性并可用于构建基于多种特定游戏类型的相机行为。这里，建议读者对各种游戏类型进行深入分析，并以较为独特的方式实现不同的方案间的有机结合。

第6章 相机脚本

相机系统设计的主要组成部分即是相机行为的动态调整或控制功能，并以此引入新的相机行为。根据游戏体验（抑或游戏事件）的变化或者玩家角色属性的变化，上述功能不可或缺。但在某些情况下，只需强调特定的环境特征或确保游戏对象交互序列可见即可。另外，非交互式电影视频序列也常出现于游戏中，进而对类似的游戏事件以及玩家行为做出适当的反馈。在大多数情况下，有必要依据环境的复杂度对相机的行为进行适当的调整，较为常见的做法是限制或规定相机的移动（或方向）。

本章主要讨论游戏事件管理需求及其相关问题、强调相机系统所涉及的细节问题并提供相应的实现方案。

相机脚本概述

相机脚本机制是指视频游戏中对象交互的定义以及控制过程，该术语与电影剧本具有相似之处。然而，电影剧本定义了一系列的非交互动作及其对应的描述方式（以手写剧本或图版的方式）。相应地，游戏设计脚本并不存在文本格式，虽然在开始阶段可采用文本加以记录。而且，尽管游戏设计脚本表述了一系列的线性事件（常用于生成类似的非交互式序列），但它更多地关注于动态变化的场景，当脚本机制用于控制基于玩家动作的相机属性时尤是如此。

虽然本章主要探讨相机脚本的定义和控制操作，但对游戏脚本机制进行简单的描述依然是必要的。在游戏系统中，对游戏事件的时间以及对象间的通信加以控制则十分常见，而环境上下文中的脚本机制定义了游戏对象间的相互关系。

复杂问题或机制的分析过程则是脚本操作的用武之地，尽管该过程实现起来相对缓慢。有时，针对程序员访问的、游戏对象的内部状态变量以及参数，游戏设计者亦有权对其进行调整。当采用脚本机制实现复杂的游戏体验以及对象行为时，应对此加以谨慎处理。例如，针对于游戏中所涉及的 AI 行为（如 Boss 战），在游戏引擎内部对其直接编程常视为最佳解决方案。相应地，游戏设计人员所提供的脚本机制也无须程序员加以干涉。

脚本类型

针对游戏系统，存在多种方式对游戏事件的脚本机制加以定义。同时，相关需求条件以及所选方案亦与游戏类型有关。例如，动作冒险类游戏与赛车类游戏之间的脚本机制则截然不同。针对前者，玩家行为所导致的动作以及问题处理方案使得相机行为千变万化；而对于后者，则首选相机方案则主要针对回放相机。

如果不考虑游戏类型，脚本系统应适用于多种需求方案。这里，较为常见的脚本方案是脚本语言和事件管理机制。

脚本语言

顾名思义，脚本语言将涉及相关文本定义，进而提供游戏事件的处理机制。需要说明的是，脚本操作并非总是通过编辑或维护文本文件而完成，某些脚本语言亦支持图形界面操作，例如有限状态机（FSM）以及可视化编程组件，下面将针对各类实现方案加以考察。

基于文本的脚本语言

针对通用编程语言以及基于特定游戏或游戏类型的自定义语言，存在多种对应的实现方式。这里，基于文本的脚本语言常采用以下两种形式，即预编译方式和解释方式，前者将生成最终的可执行机器码并链接至游戏程序中，以在必要时作为独立的线程（与特定的游戏事件关联）加以调用。当然，也可在特定时刻从外部加载这一类线程，进而降低内存开销。在解释型语言中，系统将生成数据的中间形式（基于原始文本）且并不执行于目标平台之上，这一类中间形式通常独立于目标平台并在游戏执行时转换为可执行代码（这也是解释型语言的由来）。该过程大大简化了不同目标平台之间的转换操作，但解释过程依然会产生潜在的性能开销。

通用编程语言

采用现有的程序设计语言作为游戏事件的脚本语言也是一类十分常见的处理方案，如 C、C++、Lua、Java、Fortran、LISP 等，每种方案均存在各自的优势。

- ☑ 通常情况下，相关语言应具备清晰的定义以及丰富的文档资源和相关示例。
- ☑ 存在对应的编译器以及调试器（注意：某些特殊平台可能并不包含该功能）。
- ☑ 在各功能项的实现过程中（理论上），相关语言应包含一定的灵活性，其中包括游戏引擎内部函数的访问能力。然而，某些特定实现方式将阻止相关组件访问游戏引擎，如内存管理。
- ☑ 解释型代码可能会运行于某一虚拟环境下，进而防止游戏脚本访问或修改相关数据，这将有效地避免系统的崩溃。然而，基于脚本机制的传统程序设计语言也包含某些缺陷。
- ☑ 脚本代码在游戏中运行时不存在相应的调试器对此加以支持，这在解释型系统中尤为常见。
- ☑ 游戏设计者应具备相应的程序设计背景知识，当然，某些设计人员可能会对此感到不适。游戏设计人员通常未受过任何编程训练且不具备程序的调试能力，这大大限制了他们在程序设计领域中的能力。但实际情况并未如此悲观，设计人员通常更擅长于游戏概念设计，而非具体的实现过程。因此，关于游戏的相关规范及其实现，设计人员应与编程人员保持频繁的接触。
- ☑ 脚本语言支持复杂、多级嵌套代码的编写（或在脚本对象外部调用函数），“诱惑”往往来源于此。作为一项重要的原则，脚本应尽可能地保持简洁。相应地，代码长度不应超出一页纸的范围。当然，当与通用编程语言结合使用时，上述显示条件也并非一成不变。另外，考虑到调试功能以及其他引擎功能项的访问能力，较为复杂的功能项建议在游戏引擎内加以实现。

☑ 针对内存管理以及垃圾回收机制，较好的控制方案依然寥寥无几。在内存有限的游戏机游戏以及类似的嵌入式系统中，这往往也是问题的根源所在。

自定义脚本语言

相对于主流程序设计语言，自定义风格的脚本语言则具有较强的适应性，这一类语言往往完全处于开发人员的控制范围内，并包含针对于特定问题领域的相关功能项（如语法、关键字以及操作符）。随着语言以及分析工具的不断强大，自定义语言的维护和扩展操作的管理难度将大大降低。

自定义脚本语言的早期应用包括游戏 Maniac Mansion 中的 Script Creation Utility 工具，该工具在20世纪80年代中期曾出现于一系列的计算机冒险类游戏中。针对电影场景中角色的移动以及定位，该语言提供了相应的关键字以及结构，同时还包含了传统的程序设计结构。实际上，采用 SCUMM 编写的程序甚至可直接视为脚本。SCUMM 最初的意图即是使缺乏程序设计经验的设计人员可对游戏事件以及对象交互过程（如博弈类游戏、角色的交互以及运动等）实施有效的定义和控制操作。然而，随着 SCUMM 的不断普及，其问题也接踵而来。

☑ SCUMM 难于维护、难于扩展（包括与其他平台间的接口）。
☑ 由于采用字节码解释指令，因而 SCUMM 执行速度较慢。
☑ SCUMM 调试能力有限。
☑ SCUMM 需要自定义工具用以处理动画操作等内容。
☑ SCUMM 仅包含有限的程序设计结构。

如果该语言在设计之初即经过深思熟虑，则上述大部分问题均可得到妥善解决。然而，游戏开发过程是一项复杂而庞大的任务，因而应由独立的工具开发部门对这一类辅助语言进行额外的维护，而非当前游戏开发团队。除此之外，SCUMM 自身功能也表明，该工具并不适用于程序设计任务，特别是 SCUMM 并不支持标准程序设计语言中的相关数据结构，因而需在语言复杂度以及亟待解决问题之间找到某种折中方案。如果相关问题难以处理，则标准程序设计语言依然是首选方案。

技术要点

典型的自定义脚本语言如下所示：

```
MyActor = UseActor( " Bob " ); // N.B. typeless
TheDoor = UseActor(" Door " ); // N.B. No error handling
With MyActor; // refer implicitly to our actor
WalkTo(TheDoor);
TheDoor.Open();
FaceCamera();
CloseUp(); // camera shot definition
Say(" Leaving.mp3 " ); // a resource name would be better
Wait(5);
Exit(TheDoor);
Close(The Door);
StartCinematic( " BobLeaves " ); // an identifier of some kind
FadeToBlack(2);
```

需要注意的是，该语言可能具有面向对象特征，但与大多数程序设计语言相比，脚本语言的阅读方式更类似于英文阅读。另外，脚本语言并不强调数据的类型且不包含指针，进而可避免与内存写入相关的潜在问题。这里，关键字和函数调用往往与特定的问题领域相关，且可据此对游戏对象进行直接处理。相机脚本亦不复杂，主要用于激活新增相机类型或对现有相机属性加以修正，如下所示：

```
Script Trigger.Entered(theActor)
{
    if (theActor = = " Player " )
    {
        attachedCamera = GetConnectedCamera();
        if(attachedCamera.Valid())
            StartCamera(attachedCamera);
    }
}
```

上述激活操作较为简单。相机脚本技术主要用于激活操作以及不同相机行为间的插值计算，尽管也可以对单一行为属性进行调整（如与目标对象之间的距离）。另外，高级脚本语言还包括电影视频技术的应用，如镜头的构成、场景切换以及其他相机属性。相机镜头脚本的操作方式如下所示：

```
Camera.Script
{
    thePlayer = GetActor(" Player " );
    HeadShot(thePlayer, 30, 15);
    // camera's angular displacement
    Transition = CameraTransition(" CrossFade " , 2);
    // i.e. over 2 seconds
    MediumShot(thePlayer, –45, Transition);
    // specifi es angle relative to character and
    // the transition type
}
```

虽然脚本语言具有一定的实用性，但在大多数情况下，设计人员均会在交互式环境下进行开发并以此缩短冗长的开发周期。场景编辑器（本身即可能为一款 3D 建模工具包）通常具备相应的预览功能，尽管对游戏逻辑的依赖性可能会排除这一类选择方案。

有限状态机

有限状态机可视为一类抽象的编程结构并多用于游戏领域内（尤其是 AI 循环中），进而可对生物或其他游戏对象的行为加以定义。FSM 的用途并不局限于对象逻辑，游戏系统的状态管理（如用户界面）也常采用此方式进行处理。

FSM 由以下元素构成：状态、转换过程以及行为。

☑ 状态。从本质上讲，对象的行为可分解为一系列的离散状态，各个状态体现了该对象独立的行为条件。针对于 AI 对象，常见状态可能包含攻击状态、巡游状态和躲避状态等。类似的方案也适用于博弈类游戏以及其他游戏结构中（例如门对象，其状态包括已开启状态、已关闭状态、开启过程中以及关闭过程中）。游戏事件可导致相关对象的状态发生改变，通常情况下，

状态机定义了相应的状态以及状态变化时所需的条件检测操作。

- ☑ 转换过程。转换操作确定了状态变化时的相关方法和规则，状态之间可能存在多种转换方案，因而需要进一步确定转换规则的执行顺序。针对某些较为复杂的状态机，考虑到某一对象将存在大量的潜在状态，因而对其变化进行跟踪或更新将变得十分困难。当满足预置条件时，即会发生相应的转换操作。例如，从外部对象接收到特定消息或状态变量的变化（如计时器为0）都可视为典型的操作示例。
- ☑ 行为。当出现某一状态转换时，相关行为可视为对应的函数调用操作。这里，较为典型的示例包括某一状态的入口和出口。另外，当某一状态处于激活状态时（如 Update 函数），相关行为动作也可实现为周期性调用。因此，潜在转换操作的条件测试可视为周期性行为动作并在每次更新时执行。

状态机多采用图形方式加以定义，在基于自定义方式的编辑器中，这将使得用户方便地察看状态间的关联，通过简单地设置新状态并拖曳状态间的连接标记即可定义不同状态间的相互关系。针对状态的定义以及转换操作，一些较为专业的图形工具（如 Visio 或 OmniGraffle）均提供了相应的可选方案。这里，假设状态以及连接信息均可以用方式导出（如较为流行的标记语言 XML）。典型的 FSM 如图 6.1 所示。

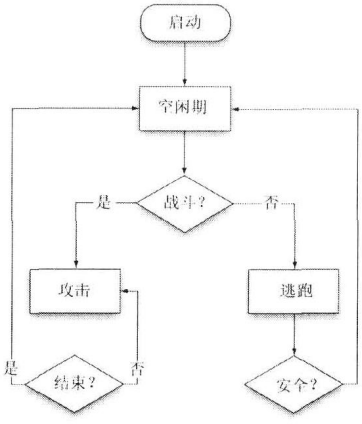


图 6.1 基于游戏对象的简化 FSM

层次型 FSM 则是一类更为高级的 FSM，其中，对应状态均可视为独立的 FSM，并以并行于父 FSM 的方式（亦即，父 FSM 可终止子 FSM）或作为自包含函数执行自身的逻辑内容。

可视化脚本语言

可视化脚本语言（或可视化程序设计）使得程序员或设计人员更多地依赖于图形界面对脚本的功能模块进行组装。通常，这可在相关界面上通过点击操作（或拖曳操作）得以实现。从本质上讲，符合当前条件的程序设计模块均被显示（如通过按钮或下拉菜单），并根据上下文环境以及实际用途进行选择。例如，当选择了某一循环结构后，仅对应的循环操作符被选取。在某些情况下，下拉菜单也可用于选取程序结构、可选数值以及枚举值。该方式将大大降低手写输入量并确保程序的语法正确性。

当采用上述方案时，程序可通过类似于文本编辑器的方式进行察看，但某些独立元素则不支持编辑功能（如关键字）。其中，数值可通过简单的鼠标单击方式进行修改。另外，可从当前位置对程序进行拖曳以修改其执行顺序。

脚本语言中的问题

脚本系统就其功能性和复杂度而言变化较大，过度的脚本方案往往会阻碍开发进程，针对内嵌于主游戏引擎中的解释型和编译型语言尤是如此，其原因在于，多数内嵌型脚本方案均缺乏相应的调试能力。除此之外，脚本系统中编程量越大，所涉及的程序设计知识也就越庞杂。因此，针对于游戏引擎的链接功能，提供相应的结构满足此要求是十分必要的。

当采用上述方案时，对象关系复杂度、时间问题以及状态管理则很容易处于失控状态。当然，业界对此类观点尚存争论。不难想象，潜在问题最终会在 QA 阶段浮出水面，尤其是缺乏相应的调试能力时。

尽管如此，某些开发人员依然在游戏引擎中成功地采用了预定义编程语言，例如解释型 C 语言、Lua 语言，甚至是 Python 或 LISP 语言。最终，交互变化的优势弥补了缺乏调试功能时的缺陷，因此这一类方案依然存在可取之处。在诸多方面，可将设计复杂度留与主流程序设计语言环境，而对于小型可控逻辑元素，脚本语言依然存在优势，特别是在降低消息复杂度以及开发周期方面。

再次强调，如果某一对象的脚本内容超出了一页纸的范围，则应考虑将相关功能项移至主游戏引擎中，至少该部分内容过于复杂。

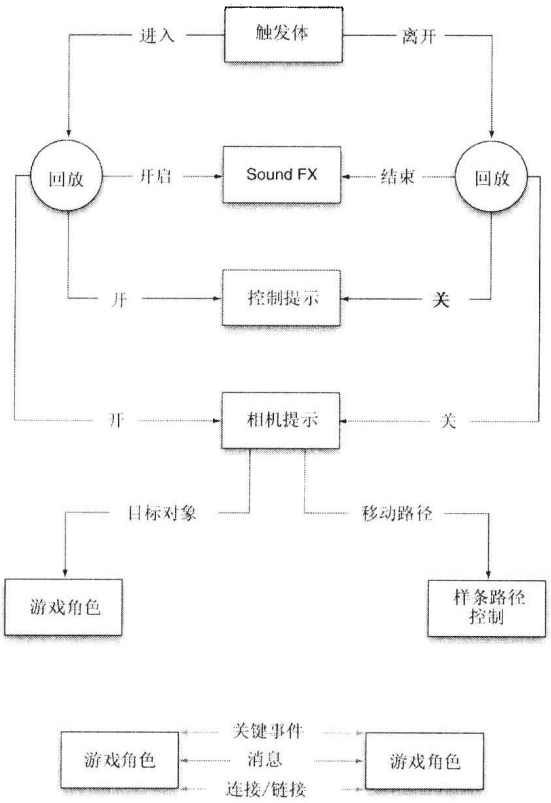
事件消息机制

基于消息的处理方案其实现过程以及用途则相对简单，基本原理可表示为：游戏对象通过某种关系而彼此关联。这里，“关系”可定义为两种信息。首先是事件信息，事件可视为一类需满足的条件集合并于随后向所涉及的游戏对象发送消息。

其次是两个游戏对象之间的相互关系，亦即，对象一确认当前事件条件有效；对象二获取该事件的通知。该关系通常采取消息机制加以实现，即数据经封装后传递至链接对象中，如图 6.2 所示。此处，定义对象间相互关系的连接有时也称作链接。

事件条件的典型示例包括触发体和游戏场景内的预置空间（如轴对齐包围盒、圆柱体等），该体空间将对玩家角色或其他游戏对象的入口和出口加以回馈。当然，体空间入口和出口的真实含义可能会随着具体内容的不同而发生变化。例如，可将玩家角色碰撞体与触发体之间的相交部分视为入口。另外，当谈及入口时，玩家角色应完全被当前体空间包围。同时，根据碰撞体中几何体的复杂度，与包围测试相比，相交测试对 CPU 的需求较低。然而，相交行为的变化很少需要如此精准的控制。

触发体的出口体现了下列情形：玩家角色处于体空间内部，且不再与碰撞体相交。针对于此，触发体应对相关状态进行维护，并可对对象表进行更新。其中，该表记录了位于体空间内以及进入和退出体空间的游戏对象。为了避免玩家重复进入或离开同一触发体，建议使用两个体空间，其边分开设置以防止上述现象的发生。此时，进入某一体空间将导致相交行为发生改变；而离开另一体空间将终止同一相机行为。图 6.3 显示了体空间的排列方式。



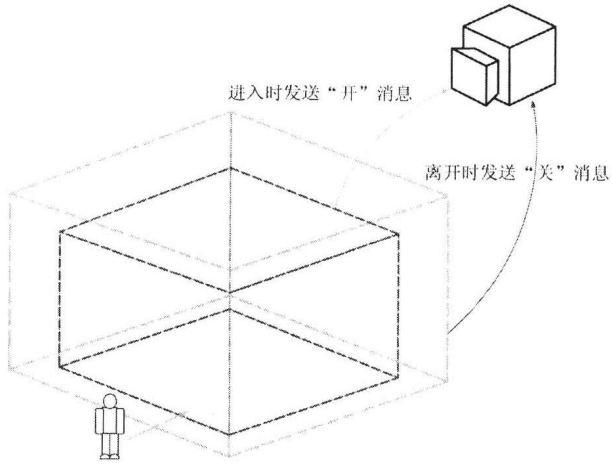


图 6.2　事件、消息以及链接之间的关系　　图 6.3　根据玩家入口或出口，用于对事件消息进行初始化的触发体

一些基于位置的事件条件包含以下内容：

☑　体空间内角色的进入与驶离（如轴对齐盒体、任意方向的盒体、圆柱体、球体等）。

☑　平面相交（如穿越通道的入口，或位于水面下方）。

☑　维护视线时对象的近似度。

位置信息并非是唯一的事件触发器，消息也可根据某一对象状态的改变进行传递（例如对象被摧毁），尽管交互序列中相机系统多针对玩家角色的位置实现反馈操作。

根据相机的活动状态，事件条件可用于向相机系统发送消息，相同机制也可用于各类游戏事件中。同时，其通用处理方案实现起来也并不困难。

一旦确定了相机行为的覆写操作，则具备了多个覆写版本以对相机行为加以操控。同时，根据相对优先权，可编写相关方法对各覆写版本加以排序。

被传送的消息常可修改目标对象的状态，比较常见的应用是游戏对象的执行规则进行初始化操作（如开启或关闭一扇门）；其他典型应用还包括令对象沿某一路径运动、激活或终止某一游戏对象等。需要注意的是，消息不必产生于脚本对象自身，且通常由代码或其他脚本形式生成。

技术要点

除了导致游戏事件发生的原始对象之外，针对于消息接收对象，了解消息发送对象的状态亦是十分必要的。原始对象在多玩家环境中十分有用，其中，同一类型的多个对象（如玩家角色人物）将使得多个脚本消息同步发送，例如玩家

角色进入触发体时所导致的相机行为的改变。不难发现，确保与正确玩家关联的相机管理器接收适当的事件消息是十分必要的。

事件消息机制应包含相关方法用于定义各类游戏对象之间的关系，且常作为游戏编辑器中的部分基础功能项。然而，考虑到连接数量众多，因而应对连接显示数量进行适当的过滤。

导致消息发送的常见事件包括下列内容：

- ☑ 启动关卡。
- ☑ 对象加载。
- ☑ 暂停游戏。
- ☑ 进入或退出体空间。
- ☑ 玩家角色遭受重创。
- ☑ 对象到达运动路径的尽头。
- ☑ 对象状态发生变化，如“对象遭受重创”、“对象被销毁”等。

常见的游戏事件消息包含下列内容：

- ☑ 开始游戏——对象开始运行其逻辑内容。
- ☑ 终止游戏——类似地，对象结束其逻辑内容。
- ☑ 激活对象——考察当前对象的渲染操作、逻辑内容等。
- ☑ 删除对象——从当前游戏中移除对象及其资源。
- ☑ 执行动作——执行对象所定义的规范内容。
- ☑ 查询状态——检查前次动作是否已经执行完毕、对特定状态进行反馈等。

技术要点

消息机制的实现过程通常与引擎相关。一类方案是使用继承后的成员函数；另外，游戏对象可通过消息管理器注册其句柄函数。由于大多数消息“类型”多采用不同的参数，因而有必要对参数传递机制加以抽象化。

脚本消息的传递可即刻执行、延至当前更新循环的结尾处，抑或根据逻辑顺序加以分组。当采用延迟策略时，考虑到相关消息可能会导致处于活动状态的相机属性发生变化，因而消息处理过程将会先于相机逻辑执行。

混合脚本方案

当然，消息机制与脚本语言之间的混合方案是可行的。实际上，大多数游戏引擎均采用此方案以提供简洁的消息机制。其中，脚本语言将生成消息并发往特定的游戏对象。与此同时，消息机制也会向游戏对象提供相应的方法并与脚本系统进行通信。针对脚本语言，一类常见的实现方法是实现一类特殊化基本对象，其“思考”逻辑（具体内容参见第 1 章）执行相关脚本内容，因此，在激活等诸多操作方面，脚本对象具有与常规游戏对象相同的操控方式。

脚本对象

脚本对象也是一类游戏对象，游戏设计者将其置入游戏场景中以生成游戏体验元素抑或非交互式

序列。脚本对象可根据相应逻辑以及与其他游戏对象之间的相互关系加以定义，抑或仅简单地定义一类对象以装饰当前场景氛围。通常，这一类对象可视为基于属性和行为的层次结构，以更好地适应于面向对象程序设计技术。其中，子对象可使用对应父对象的属性和行为，并在必要时执行覆写操作。图 6.4 显示了对象间的层次结构，其复杂度以及相关功能项也会随之发生变化。另外，游戏对象还可通过一系列的组件加以构建，进而可对相关属性以及行为进行选择，进而形成新的功能项。若不考虑实现方法，随着游戏的不断进行，游戏或相机设计者应提供相应方案以对属性或行为加以定义或实施动态调整。

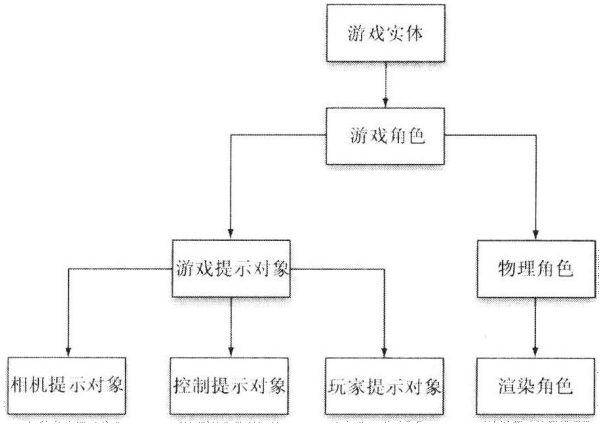


图 6.4 对象属性的层次结构

对象关系

通常，定义游戏对象之间的关系是十分必要的，同时，这一类对象并不依赖于游戏事件。上述关系常称作链接或连接。链接自身可视为定义对象关系的唯一数据，且包含单向链接和双向链接。这里，当某一对象根据另一对象定义其逻辑内容时，单向链接可用于定义二者之间的关系，或用于定义对象间特定的逻辑或渲染顺序。双向链接则通过两个对象间的直接通信实现交互对象间的信息交流。

技术要点

对象逻辑排序通常是十分必要的，该功能确保依赖于其他对象状态的对象可在对象逻辑执行完毕后获得更新，并可通过有向非循环图（DAG）加以实现。这里，可遍历对象间的一系列单向链接生成 DAG。类似地，该技术针对对象的渲染问题也是十分有效的，当对象包含较为丰富的渲染效果时，甚至会优于基于深度缓冲的硬件排序方案。

链接通常呈现为静止状态（即在游戏过程中保持不变），且常用于脚本对象访问封装于自身对象内部的外部数据。例如，在某一路径上运动的相机应与路径数据关联，另外，链接常用于对象的初始化过程中，且在运行期内无须访问其他脚本对象即可获得相关信息。然而，在某些情况下，运行期关联也是必要的，这一类关系的常见示例是相机目标对象。针对于连接相机信息，该链接用作方向目标以对游戏对象加以定义。类似的对象还可用于定义移动路径的控制点等。图 6.2 显示了状态、消息以及链接示例。

脚本游戏提示

游戏提示可视为一类脚本对象并提供了一类可能的运行期机制类型，据此，设计人员可根据不同种类的事件覆写玩家的属性、游戏控制抑或相机属性。考虑到玩家的位置、游戏事件、电影视频序列等内容，该方案可对特定的游戏体验实施有效的控制。需要说明的是，游戏脚本系统内容繁多，其全面的讲解则超出了本书的讨论范围，一些与相机相关的脚本对象包括相机提示、玩家提示、控制提示以及序列化事件管理器（可方便地实现不同游戏事件的同步操作），下面将分别对此加以分析。

相机提示

相机提示可视为一类脚本机制，该机制使得设计人员可在特定的游戏区域内对现有相机行为加以覆写或进行相应的调整。这里，基于当前相机系统的“提示”或消息可克服其自身的诸多限制条件、处理复杂的环境抑或确保特定的行为匹配于相应的游戏体验需求。其中，可对原有相机行为进行简单的调整（如相机与目标角色之间的距离、相对于玩家的位置等），抑或对该相机实施彻底的改造（如自由运动、固定路径或位置）。相机行为的改变可通过多种方式进行触发，例如游戏对象（如玩家角色）的物理位置、定时器以及其他基于事件的相关机制。

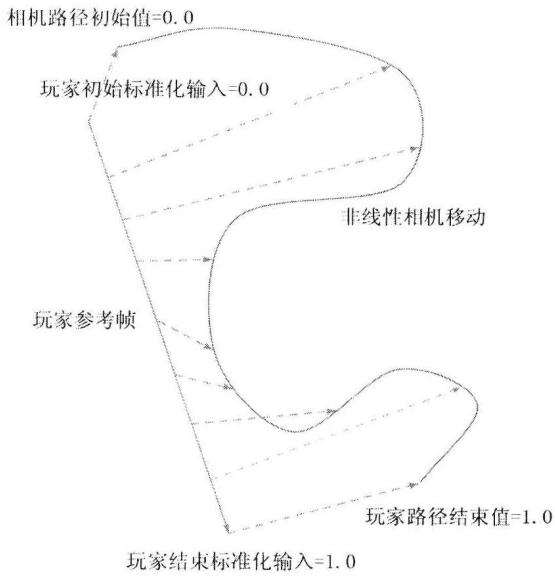
游戏脚本的深入讨论则超出了本书的范围，然而，相机提示的开启和关闭机制还是必要的。除此之外，多个相机提示同时存在也是可能的，针对于此，某些优先权控制方案则提到日程上来，这一类方案通常需要维护一个处于活动状态的相机提示信息表。另外，发送至活动态或非活动态相机的消息可能呈现为任意顺序，因而相关消息需要执行队列操作并在当前帧结尾处加以处理。另一类潜在的问题是，当游戏提示对象被视为处于活动状态时，是否存在从内存中移除或加载这一现象。类似地，这一类对象很可能在玩家进入特定游戏区域之前即被激活。

可被相机提示对象覆写的相机属性包含下列内容：

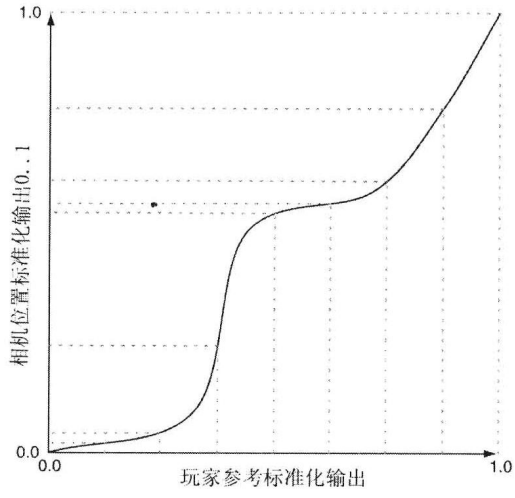
- ☑ 相机行为类型（静态、基于路径的行为等）。
- ☑ 位置和方向（如静态相机）。
- ☑ 相对于目标对象的位置偏移。
- ☑ 可视偏移/目标对象选取方案。
- ☑ 移动速度。
- ☑ 视域。

上述属性值可根据游戏对象的行为加以动态调整，其中，最为常见的属性是玩家的位置数据。通常，可依据玩家与环境参考帧的相对位置对相机行为加以调整。例如，如果玩家角色正在从事登山活动，则相机位置和方向应随玩家的攀登高度而变化（可能会在某一高度处生成全景图案）。

一类有效的参考帧是在游戏场景中定义某一路径，该路径可采用线性方式、样条曲线方式以及其他形状加以确定。随后，可通过相对于当前参考帧的玩家位置生成标准值（位于 0～1 之间），相关示例如图 6.5 所示。针对于特定的问题相机属性（如 Z 坐标），该标准值可映射至对应的值域中。这里，映射函数的类型并无明确限制，较为灵活的方式是采用 2D 控制样条（如分段式 Hermite 曲线），并将标准值映射至特定的相机属性中。在某些情况下，无须使用标准值作为控制样条的输入值，当输入值的量值可变时，其操作将更为方便（例如，若样条路径发生改变）。这一类映射函数在相机系统的诸多方面均颇为有效，当采用非标准值时，输入值与输出值之间的关系变得更易于理解。



（a）玩家与参考帧之间的相对位置可视为位于 0～1 间的标准值



（b）标准值通常映射至 Hermite 样条之上，进而生成新相机属性，如某一路径上的位置数据

图 6.5　相关示例

玩家提示

根据环境复杂度抑或游戏体验需求，不仅需要对相机行为进行覆写操作，在玩家角色的行为或操控方面同样如此。类似于相机提示，此处依然需要玩家提示对其操控以及行为特征进行临时性调整。

下列内容列举了需要覆写或通过玩家提示加以确定的相关属性：

☑ 阻止玩家的运动。通常，玩家可以自由方式移动或被限制于某一特定运动方式，具体情况则根据当前上下文环境加以确定并通过某种方式告知玩家。

☑ 对玩家进行重定位操作。在某些情况下，对玩家进行重定位操作是十分必要的（例如，修正

与环境元素之间的对齐方式，抑或在视频序列播放完毕或者进入某一新关卡时重新激活玩家）。

- ☑ 对玩家的移动特征进行调整。通常情况下，根据当前环境的变化（如淹没于水中），需要针对玩家的某些移动特征指向相应的覆写操作（如加速运动、最大速度和重力等）。
- ☑ 应对特定游戏体验进行标记。玩家提示提供了一种机制并针对特定游戏体验向游戏引擎发送消息（如溺水、Boss 战等）。

类似于相机提示，玩家提示也可通过多种方式激活或终止，甚至可根据玩家的活动状态进行适当的调整。在某些情况下，玩家提示还可用于改变玩家角色的真实状态，如将武器置入皮套中。

控制提示

类似于玩家提示，控制提示也可视为一类脚本对象，并可以实现覆写操作、关闭操作以及玩家操控的动态映射操作。针对于此，相关对象可继承于某一通用游戏提示类，且该类与控制优先权、激活以及终止等各项操作相关。此处，根据实习过程中的特殊性，必要时，控制提示可结合多项控制内容以实现覆写操作，相关事例包含以下内容：

- ☑ 关闭特定的控制或玩家操控集。此处，较好的方案是关闭映射至特定控制器按钮的玩家控制命令，而非定义特定的控制器按钮以实现关闭操作。因此，若控制映射发生变化，则控制提示依然有效。另外，这对于预置控制集的关闭操作亦十分有效，如全部运动控制、全部发射控制等。
- ☑ 确定控制参考帧以及针对新参考帧的插值时刻。在第三人称相机中，该方案通常十分有效，此处，应确保玩家控制的一致性且无须考虑相机与玩家角色之间的相对位置。这一类控制提示常与相机提示结合使用并即刻跳转至某一位置，进而使控制参考帧获取有效的调整。需要注意的是，控制参考帧应确保正确地实现玩家的意图，并使得玩家能够正确地理解控制与角色移动方式之间的关系。

多数情况下，控制提示也采用类似于游戏提示的方式实施激活、终止操作，尽管在某些时候控制提示需要从代码中生成（例如，某一 AI 对象需要临时终止相应的玩家控制操作）。另外，终止方式亦不一而同，一种较为简单的方法是采用计时器机制，当到达既定时段时，控制提示将自动终止。自然地，计时器可与其他类型的游戏提示结合使用，其行为类似于基于玩家控制的脚本机制（例如，在固定时间段内终止玩家的控制操作）。一种动态、可扩展型提示终止方案是：若玩家同时按下多个按钮，则控制操作可呈现为“分解”状态。这一类组合操作常包含按钮序列、按钮的快速按动及释放（以及按钮按下的数量和频率）等。该方案通常以调色板的方式显示控制提示对象，并可由游戏设计者进行重定义。

序列事件管理器

在既定时间间隔内，大量的游戏事件需以一定的顺序被触发，该情形十分常见。另外，多个游戏事件还可在同一游戏循环内以特定的方式被触发。针对于此，序列事件管理器可视为一类有效的解决方案，其中，消息可通过时间轴界面加以定义。亦即，对象消息可根据时间间隔进行排序，相关时间点表示为序列对象接收相应的触发信息。该方案可方便地调整消息的顺序并可对相关事件加以综合考察。另外，在同一时间轴内，同一事件还可被多次触发。

序列事件管理器常用于电影视频序列中，并针对不同事件提供统一的触发机制。除此之外，序列

事件管理器还可实现事件顺序的快速调整。

技术要点

通用提示管理器以及优先权类可用于处理多种提示信息——二者之间唯一的区别是，基于多个提示的优先权的且针对具体提示类型的比较函数，而相关代码则用于覆写或存储相应的提示属性。需要注意的是，应制定相关决策以正确地恢复属性信息。例如，所恢复的前次属性集合由其他提示对象所设置，这将给当前属性的设置工作带来混乱；另外，如果未经正确测试，这将大大增加设计的难度。这里，建议对标准的默认值集合实施恢复操作。如果数据表非空，则可采用当前最高权限的提示属性。该机制的实现过程较为直观且易于理解。同时，该方案还大大降低了当前提示属性的复杂度。

在大多数脚本系统中，对脚本消息的发送顺序进行控制是一类十分常见的情形。因此，随着新增相机提示被激活，当前处于激活状态下的提示很可能在同一更新循环中被终止。与当前处于活动状态下的提示数据相比，如果新增提示对象的优先权较低，则该对象将在当前帧中定义为非激活状态。为了有效地解决这一类潜在的依赖性问题，应累积全部激活事件与非激活事件，直至所有脚本消息发送完毕。随后，处理全部非激活态事件则相对简单，该方案可确保同一游戏帧中的多个消息得到正确的处理。除此之外，针对于同一相机提示，该方案还可方便地搜索开启/关闭消息对，并对这一类消息直接加以删除。

相机脚本机制

相机系统中的哪种元素需实现相应的脚本化操作？相机属性的脚本化范围常根据相机的具体用途呈现为不同的状态。交互式相机通常针对相机属性提供了直接控制，然而，非交互式视频序列常局限于较为基础的功能项，如位置、方向以及视域。

相机脚本方法

这里，存在两种相机脚本机制可用于调整相机的相关功能项，即预定义机制和动态机制。

预定义相机脚本机制

通常情况下，针对设计者而言，相机系统设置了预定义相机行为“调色板”以满足个人所需。如果场景编辑器（或类似工具）支持宏对象，则“调色板”的创建过程将十分简单。这里，调色板可视为预定义游戏对象的合集。在脚本语言中，上述预定义选择方案可通过库或隐式添加的脚本对象加以实现。

动态相机脚本

该类型相机脚本机制仅作用于需临时调整的相机属性。需要说明的是，该方案并非定义一类全新的相机类型，且仅对所需的相关属性加以修改，而无须考虑处于活动状态下的其他相机功能项。由于这一类方案不可影响全局功能项，因而这在一定程度上限制了可调整的相机属性。实际上，脚本机制的便捷性不可超出由提示组合方案所引入的复杂度。另外，如果脚本机制未曾单独使用，则相机行为的影响因素将变得不明朗化。

相机控制

关于相机系统，脚本机制的主要用途体现在：针对于游戏体验以及电影视频序列需求的、相机系统的临时自定义能力。由于满足玩家以及设计人员全部要求的相机系统，其开发过程难度较高，因而可针对游戏体验中的特定场景执行相机行为的覆写操作。相关示例包括可在狭小空间内使用固定位置的相机，抑或在处理某一谜题时向玩家显示特定的线索。

相机脚本机制适用于交互式环境以及非交互式电影视屏序列。需要注意的是，相机脚本机制通常与场景的表现风格无直接关系。

第三人称脚本机制

相机脚本机制多用于第三人称相机。在大多数游戏体验环境中，针对第三人称相机进行操控并以有效的方式向玩家展示游戏场景是十分必要的。第三人称相机脚本机制可能涉及全部相机行为的变化，抑或仅针对某一特殊方面加以调整。为了简化脚本机制的处理过程，主要行为变化常通过特定的预定义脚本对象加以实现。随后，相机系统将基于该行为生成相应的运行期游戏对象（如基于路径的相机）。针对通用相机属性所做出的较小调整，可考虑使用单独的脚本对象（该对象游离于主行为脚本机制的管理范围之外）。这一类脚本对象（通用相机提示对象）仅改变基础的相机属性，具体内容包括视域、焦距（若存在）、过滤或渲染效果、雾距离、玩家角色的距离或高度值（仅采用单一类型相机）等。从另一个角度讲，这需要添加额外的对象并使设计人员对多个相机对象集进行跟踪，进而实现脚本机制的操作任务。

第一人称脚本机制

当采用第一人称相机时，相机操控并不十分常见，但这并不表明可忽视操控的适宜性和辅助性功能。下列内容体现了基于相机方向的自动控制操作，其作用不言而喻。

- ☑ 斜面行进。当玩家沿平行于坡道轴向方向行进时，相机高度值的自动调整可视为一项重要的功能，这将有效地改善玩家移动方向上的视野，尽管角色自身可能会保持垂直对齐方式。这里，倾斜角通常会根据玩家角色与斜面之间的相对位置进行调整，如图 6.6 所示。注意，当玩家角色针对垂直于斜面轴方向实施重定位时，应适当地减少倾斜角度。同时，相机倾斜角的符号应与玩家角色与斜面之间的相对方向匹配。亦即，如果玩家角色的方向与斜面的下行方向对齐，则相机的倾斜角应位于水平面下方，反之亦然。鉴于玩家移动时无法实现全视角观察，因而 Metroid Prime（[Metroid02]）即采用了上述方案。需要说明的是，考虑到该方案仅定义了相机的基本高度，因而当玩家在移动过程中改变视野时，此方案依然可行。
- ☑ 跳跃动作。当采用第一人称表现风格时，传统的跳跃动作将会导致问题的出现。原因之一是玩家缺乏对于环境特征的有效判断能力，如塌陷的深坑、低矮的障碍物等。同时，多数游戏在玩家角色的跳跃过程中均对移动方式采取了相应的限制条件，例如较小的侧移量，但这依

然会改变跳跃动作的整体距离长度。在这种情况下，如果玩家可获取较好的目标视点，其效果将不言而喻。一种有效的解决方案是，随着玩家的下降，仰角也随之缓慢下调，如图 6.7 所示。需要注意的是，角度的倾斜量应在控制范围内并可在任意时刻由玩家掌控。一旦玩家着陆（或十分接近于地面），则相机应逐渐恢复至默认仰角，游戏 Metroid Prime（[Metroid02]）即采用了此方案。

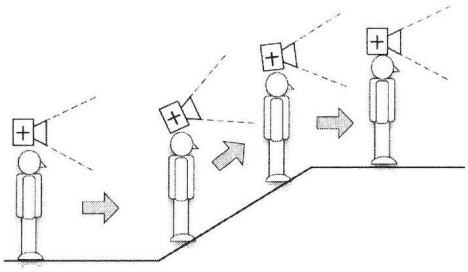


图 6.6 根据玩家位置，第一人称相机的自动仰角操作

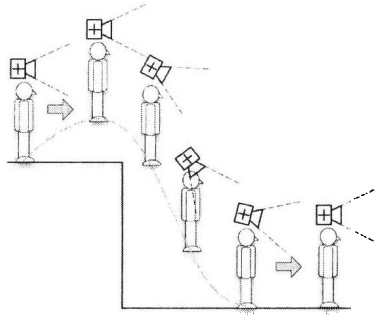


图 6.7 玩家跳跃过程中的自动仰角

☑ 画面锁定。在某些情况下，对相机方向实施完全控制是十分必要的。例如，相机系统可提供一类“锁定”功能。该功能可使玩家相对于另一游戏对象或位置以一种半自动盘旋方式进行移动。同时，侧向移动控制将使移动方式呈现为一种围绕“锁定”位置的弧状效果。在运动过程中，相机方向应自动跟踪该锁定位置。多数时候，无须同时锁定相机的偏角和仰角，尽管在第一人称相机中玩家的 2D 方向与相机的方向通常会保持一致。因此，当采用该方式锁定相机方向时，依然需要上述方案的支持。针对于此，当玩家相对于锁定游戏对象运动时，则可使用“全视角”方式进行瞄准操作。图 6.8 显示了基于第一人称相机系统的锁定效果。

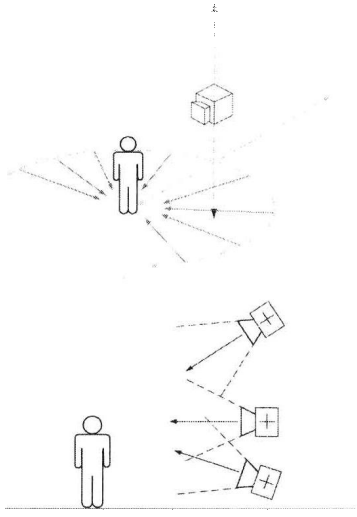


图 6.8 画面锁定功能可仅限于相机的偏角，从而使玩家可实现“全视角”功能

非交互式电影脚本

对非交互式电影视频序列进行控制可视为相机脚本的另一项重要功能。与交互式序列相比，出于表现风格的一致性考量，此类视频序列通常仅涉及相对简单的脚本机制。在不失简单性的前提下，游戏电影脚本机制可通过相机在一段时间内的移动和重定向操作加以实现，这一点类似于真实的电影拍摄技术。另外，任何影响场景渲染效果的相机属性均可通过相机脚本加以控制，如视域或景深效果（即焦距）。

此类电影视频序列多采用源于游戏内部相机提示对象的独立脚本对象加以实现。这里，电影视频相机的主要移动方式均源自游戏内部相机的移动方式，而前者多采用流逝时间值作为移动控制输入值的决策项。另外，相机之间的切换操作通常可通过开启一个视频相机脚本对象得以实现。

经验方法

如不考虑脚本机制的实现方法，某些较为实用的经验规则将有助于相机脚本机制的实现。

采用奥卡姆剃刀原理

该原理表明，设计过程应尽量采用最为简单的脚本方案。当在复杂相机行为（即具有较强的灵活性但难于定义和实现）与简单但缺乏一定灵活性的方案之间进行考察时，简单且具备一定可扩展性的方案通常可实现快捷的调整操作。除此之外，当对项目中前期定义的现有脚本内容在此访问时，简单方案更易于实现游戏体验的更改、替换操作。当对此实施进一步考察时，还会得出以下结论，即应尽量保持相机移动的简单性。相机移动方式越加负责，则玩家越有可能失去方向感（皆因这一类方案均会产生显著的方向变化）。

采用多种方式触发相机事件

简单地采用触发体并不能完全反映全部事实真相，玩家模型可能会改变相机的脚本需求条件。玩家相对位置（此处作为触发体）也可用于修改既定相机提示对象，例如，图 6.9 显示了两种相机提示对象，且均链接于同一触发体，各相机提示对象均表示为静态相机且用于跟踪当前玩家。在该例子中，当玩家进入触发体时，应选择与现有相机视觉方向实现最佳匹配的相机提示对象。

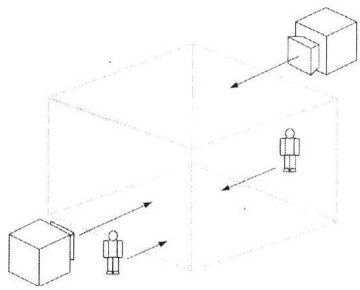


图 6.9　链接于同一触发器的两个相机提示对象，且相关对象将根据前次相机位置关系加以选择

1．支持相机移动和方向预览机制

在可能情况下，相机设计者应在游戏脚本环境下或游戏内部预览潜在的相机脚本机制。这在定义激活态相机切换操作时尤为重要（如插值计算），且仅可在游戏引擎内部加以测试。

2．在设计早期评估相机脚本机制

理想情况下，该过程应在游戏艺术设计工作完成之前或者评估操作应用之前加以执行。某些游戏开发团队将会先期开发游戏的简化版本以形成游戏体验需求，并于随后创建适宜的艺术形式。通过在早期阶段确定相机的需求条件，设计者则有机会对当前环境进行调整并较好地与相机系统进行匹配。

3．提高脚本机制的开发效率

通用相机脚本机制以及特定的电影视频序列脚本机制其开发过程具有一定的重复性。因此，考虑到调整工作对开发周期的影响以及目标平台上的查看能力，高效的开发工具是一项十分重要的内容。

针对于此，游戏场景编辑器在预览功能的改善以及扩展方面起到了不可替代的作用。

4．模拟工具行为

这里，不应要求美工师（或设计人员）在游戏工具集内学会多种3D环境遍历方式。当然，大多数用户均可在各种建模包或编辑器中进行自由切换并展开高效的开发工作。然而，统一的界面以及对工作方法的熟悉程度可大大地提高开发效率。同时，相关工作还包括术语的统一。在大多数情况下，开发环境可通过建模软件包中的插件功能得以实现，因而可保证美工师对当前任务的最大熟悉程度，并在一定程度上减缓开发任务所带来的种种压力。

5．可对游戏体验进行适当的调整

在多数游戏中，可根据游戏状态的变化对场景环境作出有意义的改变（例如，处于开启状态的一扇门随后被锁住）。因此，相机脚本方案应允许原脚本内容中出现上述变化，抑或包含不同的事件触发脚本方案。针对于相机系统的其他建议，第4章中的相关内容也对此有所提及。

6．脚本工具

相机脚本在很大程度上取决于工具集的使用，并可在游戏场景内实现相机的定义和定位。在将脚本内容转换为游戏引擎所支持的格式时，还会使用到其他工具，甚至会涉及编译过程。一旦脚本内容运行于游戏引擎中，应对其内容进行相关分析以确保执行正确的功能项。下面将简要介绍基于相机脚本的相关工具。

（1）场景编辑器

当开发视频游戏时，场景编辑器可视为一类重要的开发工具之一。在某些场合下，场景编辑器可预览全部游戏逻辑内容，但情况并非总是如此，尤其是目标平台与开发系统之间存在差异时。尽管如此，场景编辑器依然可视为一类优先选择方案，中间件软件包将对游戏开发过程提供最直接的支持，抑或通过相关软件包对现有3D设计作品进行修改。理想状态下，可自行编写插件以在编辑器内预览某些相机功能项。除此之外，针对于真实游戏环境内的浏览操作，对应工具应可快速下载相关数据信息至当前目标平台。

不难发现，除了确定消息传递机制、状态转换、事件触发以及对象间的关系等内容，场景编辑器还应提供脚本对象的定位以及操控机制。另外，场景编辑器还应支持独立脚本对象的编辑功能，并将脚本对象及其相互关系进行分组，以方便对公共游戏功能项实施复制操作。需要注意的是，场景编辑

器可能不支持脚本语言的编辑功能，但应对其用途进行适当的定义。

（2）目标平台和 PC 平台之间的通信

作为调试以及开发过程中的功能之一，游戏状态信息间的通信不可或缺。这里，假设目标平台独立于其开发环境。当针对 PC 进行游戏开发时，目标机器常定义为开发平台（针对于此，甚至可将网络 PC 用于目标平台）。有时，该通信机制可通过简单的并行接口加以实现；而在某些情况下，也可通过局域网执行此项功能。如果不考虑最终的实现机制，当执行游戏对象的逻辑内容时，其状态查询功能将十分有效。另外，与活动提示对象相关的信息及其属性也是一类重要的数据。

（3）消息/事件日志系统

维护活动事件以及事件发送消息列表是一种较为有效的调试技术。

如果可对更新循环计数器或事件的时间间隔值加以记录，则事件可按照时间顺序进行浏览。理想状态下，事件日志应存储于开发主机中，以备日后查看。必要时，事件日志可采用二进制格式加以编写，这可在一定程度上节省内存开销。然而，多数时候，从游戏中导出的日志文件应采用可读格式，且相关数据也可通过逗号分隔，从而简化了数据表或类似程序的导入操作。

（4）对象属性调试

当游戏处于允许状态时，游戏对象属性的检测功能可视为一类重要的工具。对此，程序员可访问程序调试器以对相关内容进行查看，而开发团队中的其他人员则往往束手无策。理想状态下，应可在场景编辑器中对游戏进行查看。一类较为常见的情形是，当浏览处于活动状态的游戏对象时，编辑器应暂停其执行过程。通常，这一类调试往往内建于游戏中，并可在目标显示设备上显示相关的属性值。然而，鉴于显示设备、内存空间等诸多限制条件，该方案在实际应用时可能会导致问题的出现。

（5）回放

采用既定方式对部分游戏内容加以回放可视为一项重要的调试工具。较为简单的方式是采用 VCR 对游戏进行回放，多年以来，该方式常被 QA 等部门加以采用。

对于 QA 部门而言，游戏体验测试过程及其记录工作不可或缺——如果 QA 部分缺乏这一步骤，则应从自身角度寻找原因。另外，录像带（甚至是硬盘、DVD 以及录音机）等装备通常价格低廉且易于使用，当出现演示对象的错误行为以及某些难以复制的设计 bug 时，这一类工具通常十分有效。视屏录像常可与调试输出结合使用，进而可实现相应的内部游戏状态信息。

在实际应用中，游戏体验还可通过捕捉控制器输入数据这一方式加以实现，并于随后将此类数据替换原有的控制器输入。这里，假设捕捉点处的游戏状态可实现有效的复制。此处，应确保不确定因素——例如随机数（通常为伪随机数）——一并被复制。当采用上述方式时，某些因素的复制结果依然会被遗漏（如网络流量状态或延迟效果）。总之，出于调试目的，确定的游戏状态复制结果是一项十分重要的数据内容。

（6）控制台窗口

一种更为通用的调试技术提供了目标平台显示设备以及远程主机（如有别于目标平台）的文本消息打印方案。该方案可通过脚本对象（使得设计人员可调试其自身逻辑内容）或代码方式加以实现，其中，游戏引擎内部状态信息将得以显示。

为了有效地防止文本输出内容过量，应对文本显示消息做适当的限制。通常，这可通过基于信息显示类型的特定变量加以实现。有时，相关调试变量可通过游戏内部的菜单（调试菜单）或开发机器上的输入控制窗口加以设置。针对于后者，当游戏引擎处于运行状态时，开发人员可向其发送相关命令，这也令该技术显得十分强大。其中，可对相关命令进行适当的定义，进而可调整对象状态、修改

调试信息、替换操控配置信息等。根据对象的当前状态，上述方案还可用于检测特定的游戏对象，甚至在执行其逻辑内容时修改相应的属性信息。

（7）脚本调试

相机脚本调试机制涉及对脚本系统所采用的逻辑内容的理解，进而可在游戏执行过程中激活或调整相机的行为。因此，该过程使用了调试其他游戏系统时的诸多常见技术。然而，由于脚本系统用户在访问目标机器状态时往往会受到诸多限制，因而提供额外的辅助信息以显示脚本系统的内部状态是十分必要的。

下列内容显示了某些游戏脚本调试过程中的常见方法（且并非是程序调试器中的内置功能）：

- ☑ 脚本语句的执行/过滤。对于任意游戏，脚本（或由相关过滤机制定义的、当前处于激活状态下的脚本对象的子集）显示了各条执行语句的历史记录，从而可替代全功能的调试器。鉴于该技术体现了调试器中的大部分功能项，当采用编译型语言时，其实现过程并非易事。
- ☑ 调试消息日志脚本对象可生成调试文本消息，进而可显示某些内部状态。这一类消息可独立显示，亦可转换至主机之上，或者作为游戏的主要图形输出内容加以显示，抑或加以存储以备后用。
- ☑ 消息过滤机制对象间的通信可通过前述调试信息日志功能加以跟踪和显示。此处，应提供相关功能以对显示消息类型进行过滤，进而简化分析过程，并降低所生成的潜在信息量。

通常情况下，过滤机制将根据源/目标对象类型、消息类型等内容对相关消息加以限制。

- ☑ 对象状态对象类型常可针对于其行为反映内部状态信息。这一类反馈信息针对相关逻辑的执行方式颇具参考意义，在某种程度上，甚至可替代真实的调试器功能。相机提示对象可显示其在游戏场景内的方向和位置数据、潜在的移动限制条件等。第 11 章将针对某些特殊的相机调试技术进行详细的讲解。
- ☑ FSM 状态变化针对采用有限状态机的对象而言，除了用于确定状态转换的测试条件结果，显示对象的当前状态同样重要。与状态变化相关的内部变量（如计时器）同样可采用该方式加以处理。当信息以文本方式加以显示时（如通过控制台调试窗口），游戏与用于定义 FSM 的原编辑程序之间应具备一定的通信功能。针对于对象的当前状态及其在 FSM 中的位置，该技术提供了二者之间更为直接的修正方案。
- ☑ 动画状态和事件管理大多数游戏均对动画采用了混合操作以及插值计算，因此，跟踪当前动画状态以及动画间的组合方式将具有重要的现实意义。某些动画系统允许游戏事件在动画执行过程中的特定时间点被触发（如脚步所产生的音效）。跟踪这一类事件可通过更为通用的消息日志方案或在动画系统内部加以处理。

本章小结

根据游戏体验或非交互式序列中的触发事件，本章探讨了基于相机行为控制的多种可能方案。尽管多数方案运行良好，但仍可采用脚本机制对关系数据加以定义。游戏对象常用于定义新增相机行为类型，并通过既定的脚本方案加以初始化。需要注意的是，系统应确保既定脚本方案适用于游戏所需的用途类型，并可提供强大的调试能力。

第 3 部分：相机工程

一旦全面理解了实时相机的设计需求条件，读者可着手处理相机的实现方案。本部分内容将详细讨论游戏相机的各项属性、内部设计、功能以及实现方案。

第 7 章　位置和方向

相机在游戏场景中的定位方式包含以下两项内容，即位置和方向。相机的首选位置——即期望位置——定义了游戏场景中的位置数据，且据此位置可生成视见内容，相关内容应满足一定的美学以及游戏体验需求并可实现动态改变。与此相关的问题是，相机如何到达期望位置，这其中包含了物理考察因素以及必要的限制条件（如避免穿越环境表面或其他游戏对象），该过程也称作相机的漫游。针对当前位置与期望位置之间的相机漫游以及真实移动方式，第 8 章和第 9 章将分别对此加以阐述。

相机的方向则定义了游戏场景内相机的朝向（可能还包含基于视见方向的相机旋转操作）。需要说明的是，二者之间彼此关联。首先是期望方向，该术语体现了基于期望位置的一类必然结果且取决于美学要求以及游戏体验变化。其次是相机与期望方向之间的重定位操作。本章将对此进行深入讨论。

坐标方案

针对于全部游戏相机类型，相机的位置和方向属性定义了最为基础的数据信息，无论是交互式相机亦或是视屏相机皆是如此，且与场景画面的表现风格无关。在多数情况下，相机属性的定义方式皆与游戏对象所采用的参考帧相关。该方式常称作世界空间或世界坐标系。独立的游戏对象可根据自身的位置和方向定义其坐标，即局部或对象空间。该方式常用于确定游戏对象的顶点，进而形成该对象的三维表现形态（或模型）。

这里，每种坐标系统类型均由 3 个轴构成且轴间彼此呈直角，并从公共交点处（即原点）向外延展。另外，各轴的名称随游戏引擎或建模软件包的不同而发生变化。美工师、设计人员以及程序员在其各自领域内均已对“轴”这一概念达成共识，这一点十分重要。从本质上讲，为了避免不必要的误会，相关人员应对此持有相同的见解。

下面将考察相机系统中最为常见的坐标系统。

世界空间

世界空间坐标在坐标方案中较为常见，坐标值定义了游戏场景中的绝对位置。渲染过程将坐标从世界空间转换至相机空间，并于随后再次转换至二维屏幕空间。

相机（视见）空间

当谈及相机空间时，相机通常位于坐标系统的原点处。因此，相机的前视方向（由相机位置处延伸）一般对应于某一坐标轴。取决于坐标方案所采用的定义规则（即左手规则或右手规则），前视方向

通常为 Y 轴或 Z 轴。

屏幕空间

屏幕空间内的计算结果通常可通过相机空间坐标经投影矩阵转换得到。针对屏幕空间坐标而言，令其原点置于输出设备的中间位置则是一类十分普遍的做法。此时，X 和 Y 坐标在-1～+1 范围内向外延展（相应地，负值延伸至屏幕的左上角）。同时，Z 坐标则体现了屏幕前、后方位置，再次强调，该值位于-1～+1 范围内。需要注意的是，屏幕空间的边界对应于视锥体投影的界限位置处，因而会呈现包含非平行面的空间体（透视视图变换所产生的对应结果）。

局部（角色）空间

在该坐标系统中，玩家角色（或其他游戏对象）将位于中心原点位置处且坐标轴与当前角色匹配。

对象相关

在该坐标系统中，相机位于原点位置处，且坐标轴取决于对象与相机之间的相对位置。从本质上讲，该坐标方案随着相机以及对象的移动呈现为动态变化，且相机朝向可能不会与前向轴对齐。如前所述，该方案常用于计算控制参考帧（更多内容可参考第 4 章）。

上述不同坐标系统可用于定义相机的属性，并于随后确定相机的最终位置和方向。例如，在某些情况下，需要定义相机相对于玩家角色朝向之间的期望位置（即角色空间）。在该例子中，偏移值将从角色的局部空间转换至世界坐标空间内，并于随后确定相机的移动方式。

期望位置

当考察相机的移动时，重点内容是相机的最终位置。该位置常称作相机的期望位置或目标位置（尽管后者可能会与场景特定位置处相机方向的确定方式相混淆）。相机的定位和定向方式决定了相机自身以及玩家对场景世界的观察方式。错误的定位或运动方式将导致玩家的观感满意度急剧下降甚至会绝然地退出当前游戏。

这里，相对于玩家角色以及目标位置，维护适当的距离和朝向可视为相机移动时一类重要的考察方向。在某些情况下，上述目标位置可能会与期望中的平滑移动产生冲突，当目标位置快于相机时尤是如此。针对第三人称相机，定位方式应保证玩家流畅的视觉画面以及环境场景的正确展现，这一点同样重要。

另外，相关定义方式还将根据相机的表现风格发生变化。下面将分别考察第一人称相机和第三人称相机。

第一人称相机定位方式

针对大多数第一人称相机，期望位置与玩家的眼睛位置之间通常保持固定关系。亦即，游戏场景中的某一位置接近于玩家角色对当前场景环境的观察方式（但不必是真实的头部位置，前述内容已对此有所讨论）。

当玩家角色在崎岖地形表面上行进时，应避免不必要的垂直移动。此处，在对应轴上添加少量的阻尼系数即可有效地缓解该问题。

另外，该期望位置（相对于玩家角色）通常保持恒定，即使视见朝向处于玩家操控之下亦是如此（即自由视角）。不可否认的是，围绕玩家位置的自由视角体现了相机相对于角色的真实移动方式，而非重定位操作。与此对应的例子是对层次骨骼结构进行模拟，其中包括颈部和躯干。与简单地旋转相机相比，当角色以正常姿态行进并采用全视角时，这将提供大范围的移动方式。

在第一/第三人称相机混合方案中，相机常处于玩家角色的外部附近位置处（传统的第一人称相机位置），针对于瞄准操作以及全视角观察方式，这将提供较好的视觉画面（例如角色行进至墙面时的回退行为）。一旦自由视角过程或战斗结束，相机将再次回退至相关位置处。另外，无论是第一人称相机还是第三人称相机，应尽量避免玩家行进于崎岖表面时所产生的震动现象、与场景环境抑或其他游戏对象之间的配置现象等。

某些游戏还将会引入电影视频序列，其中，第一人称相机的位置（通常还包括方向数据）可能会脱离于玩家的直接控制。当遍历复杂场景时，该方案减轻了玩家的操控“负担”，并采用电影视频的方式展现当前场景，尽管这会在一定程度上降低玩家的临场感。该情形多出现于瞄准操作以及其他受限型交互操作。与受限型相机技术相关的案例包括游戏 Panzer Dragoon（[Panzer95]）以及街机游戏 Virtua Cop 系列（[Virtua94]）。

第三人称相机定位方式

如第 2 章所述，根据相关定义，第三人称相机将位于玩家角色人物外部某一位置处。在多数第三人称环境下，相机位置通常参照玩家角色方向加以定义且多位于玩家的后上方位置处，进而提供具有一定高度的开阔视野。除此之外，多数第三人称相机均需要定义某些特定行为以适应复杂游戏场景以及相对特殊的游戏体验，进而展现具有最佳效果的游戏画面。下面将考察 3 种相机定位方式，即自动方式、玩家操控方式以及混合方式。

自动方式

针对于相机定位，某些游戏提供了自身的控制方案，从而将玩家的操控能力排除在外。某些时候，该方式可确保相机完全位于游戏场景范围内；但通常情况下，相机的自动模式仅为了减缓玩家的“负担”抑或调整单一的相机操控模式而已。当然，相机的自动方式也会产生某些负面影响，例如，玩家行为将受到一定程度上的限制作用。另外，该方式还应确保相机定位机制可满足多种游戏体验。

玩家操控方式

针对于相机位置，第三人称相机常允许玩家参与一定程度上的控制操作。当然，控制程度将会受

到移动的制约，例如采用预置相机选取方案（如驾驶类或赛车游戏），或简单地限制物理偏移量。某些游戏则对玩家的相机操控不采取任何限制，例如多玩家在线游戏，且全程支持游戏机场景环境内甚至是外部的相机控制操作。当然，该方式并不值得鼓励，至少玩家不应采取此方式对相机实施移动或重定位操作，以避免产生不合时宜的游戏场景画面。可以肯定的是，相机应被限制在游戏场景内部（也存在某些例外情况，具体内容可参考第 4 章）。针对玩家与相机定位之间的直接控制操作，不应忽略。针对相机与玩家之间的视线缺失问题，一类简单的处理方案是对相机实施某种限制，进而避免该情形的出现。

混合方式

一类较为常见的解决方案是，相机定位方式中的大部分内容可采用自动模式加以实现，且仅支持少量的玩家操控模式。如玩家可围绕自身旋转相机位置，进而提供周围区域的常见画面，一旦控制操作终止，则相机可执行自身的重定位操作。

期望位置的确定方法

下面将考察几类较为常见的、与相机期望位置相关的定位方法。具体的选择方案通常还与游戏对象的脚本相关，例如相机提示对象（具体内容可参考第 6 章）。必要时，相关选取方案还应根据玩家的行为动作进行自动选取（例如，玩家执行预置动画视频序列、翻越障碍物以及爬梯等动作）。另外，期望位置还与玩家当前状态有关。根据定义，第一人称相机的期望位置常接近于玩家角色的眼部位置；而第三人称相机则通常与玩家角色之间具有一定的位置间隙。这里所列举的方法包含了大部分情形，针对于展现游戏最佳状态的、期望的相机行为，混合方案不失为一类有效方案。

多数时候，期望位置体现了相机的运动类型，进而可作为一种相机特征的区分方法。需要注意的是，期望位置的决策方案以及朝向该位置的移动方式应与定向方案分开考察，这一区别十分重要，尽管后者常依赖于前者。本小节中的大部分内容主要集中于第三人称相机，其期望位置与玩家期望位置截然不同，且常取决于当前游戏的体验环境。第一人称相机采用了一种“伺服”期望位置，其水平位置与玩家水平位置之间将保持一致。亦即，相关变化仅出现于垂直方向上的世界坐标轴上。针对于期望位置的实现方案，一类较为有效的方法是采用基于约束条件的相机系统。实际上，针对特定的游戏体验集，可对连续的约束条件进行适当的分布，从而获取最佳的位置数据。相关约束条件包括高度值、基于角色方向的相对角度偏移、距离等。

这里，期望位置无须直接移至某一新位置，且该新位置通常由角色方向以及其他环境因素加以确定。在大多数情况下，可根据弹性系数以及基于速度的阻尼系数对期望位置的移动速度加以控制。因此，这也不可避免地在期望位置的移动过程中引入了延迟现象。根据此方式，可针对首选位置以及最终的期望位置分别加以更新。期望位置仅体现了相机的“建议”位置，而非相机的当前位置。另外，相机不应与期望位置进行直接关联，除非期望位置的决策方案中包含了阻尼功能以及过滤机制，进而可有效地消除角色移动过程中所产生的“抖动”现象（该现象也可能由玩家所引入）。由于目标对象的任何移动均会即刻影响到全部游戏场景画面且与阻尼效果无关，因而相对于移动目标对象，相机直接移至某一固定位置常会导致画面失去自然感。

这里，存在多种方式用于描述相机的期望位置，且与特定游戏的游戏类型相关。然而，在单玩家游戏中也会存在某些偶然现象并需要根据上下文环境对期望位置进行调整。下面将考察某些较为常见的决策形式。

静态相机

无论如何，静态相机可视为一类最为简单的移动处理方式。静态相机包含以下两种风格，即固定位置以及动态固定位置。需要注意的是，其他对象也可由静态相机加以支配，进而实现游戏场景的漫游功能。因而，静态相机的真实含义是指相机自身并未发生任何移动。

固定位置

固定位置相机呈现为静止状态，且可实现自由的重定位操作，并在必要时朝向其观察位置。当然，重定向操作的范围（水平方向或垂直方向）、角速度等内容可能会存在某些限制条件，具体内容可参考第 11 章。通常情况下，该类型相机多用于某些特定的封闭游戏场景，相机的移动常会产生混乱感以及迷失感。另外，当空间过于狭小且不足以在相关位置架设相机以实现对角色的有效查看时，抑或相对于相机的平滑移动，碰撞几何体过于局促时，这一类情景皆是该类型相机的用武之地。某些固定位置可根据游戏的戏剧性特征或者其他美学条件加以选择，进而强调特定的场景特点，抑或防止玩家观察到某些不完善的几何体对象。

该类型相机的一个缺陷是，目标角色可能距相机过远，以致玩家难以查看到相关角色的行为动作；或者，角色人物可能会被当前环境所遮挡。除此之外，玩家角色还可能移至相机的上方或下方，该问题不仅局限于静态相机。当发生上述情形时，相机的方向将会迅速发生改变（也称作垂直扭曲变形），特别是相机沿其前向向量旋转时，这将导致玩家失去原有的方向感。此时，相对于相机的控制参考帧将变得毫无意义，进而加重玩家的混乱感。该问题应及时得到处理（具体内容可参考第 10 章）。

同时，这也可视为与控制参考帧相关的一个潜在问题。第三人称相机则趋向于采用与相机或屏幕相关的控制操作。针对于屏幕相关的控制操作，由于视野画面自身即表明了当前控制含义，因而静态相机通常不会对问题有所展现。针对第三人称相机，角色相关的控制操作则会导致问题的出现，当然，相关问题多发生于固定位置相机。同时，该问题与“无线操控车辆”问题具有相同的性质。从本质上讲，当角色转向并朝向相机位置时（亦即，角色的前向向量逆转且平行于相机与角色之间的向量），则控制参考帧也会根据玩家的意图加以逆置。例如，向左推动手柄将导致屏幕相关的运动呈现为右移状态，反之亦然——这与期望中的结果相反。在相机系统中，若角色禁止朝向相机（鉴于位于玩家角色“后方”的相机重定位操作），则应避免上述内容的出现。

关于控制参考帧及其特定问题的处理方案，读者可参考第 4 章以获取更多的内容。

动态固定位置

动态固定位置相机从本质上讲可视为一类镜头相机，其中，相机的初始位置并不通过游戏场景中的静态位置加以定义。相反，其位置通常由某些外部资源加以确定，较为常见的情形是采用任一处于活动状态下的、相机的当前位置定义位置数据。该情景类似于：当相机处于活动状态时，即刻将该新增相机“降入”游戏场景中。某些飞行类游戏常采用此功能，并使得玩家可定位于外部相机视野画面处（例如《游戏星球大战：帝国的阴影》[Star96]）。其他方式则根据特定方向时目标对象的取景、屏幕

尺寸等内容对位置数据加以定义。在上述例子中，常需要在当前相机与新位置处的相机之间实施相应的切换操作，该操作通常可采用插值计算，或简单地通过跳接镜头加以实现。

该类型相机的一种可能用途是，当玩家从高崖处跳下时，相机保持静止并对处于下落状态下的玩家进行跟踪；当玩家着陆或接近于地面，将再次执行重定位操作。

如果当前相机替换为静态相机，则新相机需要使用到当前活动相机的位置和方向数据。根据相机切换点处的速度，上述方案将导致相机画面处于突然停顿状态；更为常见的情形是，为了便于切换操作，相机将短时期内处于锁定状态。因此，有必要通过插值计算减轻相机切换时画面变化所导致的不适应感。

伺服/跟踪操作

伺服（或跟踪）相机常与游戏场景中的其他控制对象协同移动，相关对象是否为玩家角色则需要视具体情况而定。相对于主对象，相机可能被锁定于某一位置处；或者，根据相关需求，控制对象仅描述相机的期望位置。

伺服相机可视为一类较为常见的第三人称相机类型。在大多数游戏中，相机的位置往往取决于玩家的位置。当然，伺服相机可从“主对象”处获取相应的方向和位置数据。此时，相机简单地获得玩家的某一固定位置，该位置通常为玩家的眼部位置。如前所述，这一类相机常称作第一人称相机。

针对相机方向与玩家角色方向，二者将常呈现为同步状态，仅在某些特殊情况下，相机由玩家直接操控。另外，控制操作多与玩家角色的基础方向相关，即自由视角行为，相关内容将在第 11 章加以讨论。

当玩家在场景环境中遍历时，其眼部位置可能会在垂直方向上出现少许变化（该位置源于角色相关位置数据）。针对于此，可在眼部位置移动时添加少许垂直方向上的阻尼效果。另外，相对简单的低通过滤器即可实现相机的平滑移动。当玩家受外力驱使运动时，阻尼效果不可随意添加，例如玩家位于电梯内或移动平台时（且未发生碰撞）。除此之外，如果玩家角色可穿越或接近经渲染后的多边形网格表面（如水流表面），则应对相机位置加以控制，以避免网格与视锥体近剪裁面之间产生交叠。一类处理方案是保持相机的垂直位置直至角色完全清晰可见（或完全潜入水中）。此处，可在垂直分量上添加阻尼效果，以保证相机处于正确的位置处。需要注意的是，此时，相机依然会呈现为不连续运动状态，但与经视锥体作用后的、表面多边形中清晰可见的剪裁边相比，该效果已然获得了可喜的进步。

需要强调的是，本小节所描述的定位方案（路径、表面约束条件等）均可视为伺服相机。亦即，相关方案中的基础位置可伺服于另一对象。该技术十分强大，可使设计者对相机位置加以操控进而匹配于动态游戏需求环境。一类较为常见的例子是（如游戏 Metroid Prime 以及后续篇），在旋转谜题时，可适当地旋转相机以与处于变化状态下的相关对象进行匹配。

另外，伺服相机的相对位置（或偏移量）可通过多种方式加以确定。下面将考察某些较为常见的形式。

基于世界坐标系统的偏移量

常见期望位置可根据主对象定义为世界坐标系统中的偏移量，且与对象的方向无关。当角色不存

在明显的前向方向时，抑或相机与当前角色之间需要保持相同的相对位置时，该方案则十分有效。

技术要点

考虑到无须执行旋转操作，因而偏移量通常较为直观：

```
Vec3 desiredPosition = GetTargetObject()->GetPosition() + mOffset;
```

这里，假设目标对象返回一个位置以构成偏移值的基点；在某些情况下，目标对象可使用动画模型以获取该位置数据，但多数时候，可采用目标对象的原点以避免不必要的相机移动。有时，还可使用碰撞包围体的中心点，但应防止对象动画导致该位置出现波动现象。

基于世界坐标系统的角偏移量

类似地，基于世界坐标系统的角偏移量可（最多）定义为 3 个角分量（基于世界坐标轴而非角色方向）以及一个距离值。通常情况下，只需定义两个角度即可，即仰角和偏航角，其中距离值构成了三角形的斜边。这里，旋转延迟可用于减缓相机围绕其目标对象的重定位操作，进而可确定期望位置。需要说明的是，这一类基于角度值的相机并不常见，较为典型的示例是相机在某一较高位置处俯视当前角色，如弹子球模拟器。这里，角度值将根据玩家位置或某些外部影响因素发生变化。例如，当玩家到达游戏区域的边界处时，可适当地调整相机垂直方向上的高度值。

技术要点

针对于偏移值，此处假设偏航角位于(0..2*pi)个弧度范围内，且围绕世界坐标系中的 Up 轴（此处为 Z 轴）呈顺时针旋转方向。另外，仰角则位于(0..2*pi/2)个弧度范围内，如下所示：

```
Vec3 offset(0.0f, cosf(pitch) * distance, sinf(pitch)* distance);
// construct a rotation matrix around the world up-axis
Mat3 rotation = Mat3::ZRotation(yaw);
mOffset = rotation * offset;
Vec3 desiredPosition = GetTargetObject()->GetPosition() + mOffset;
```

基于世界对象

该类型相机根据主对象和某一单独位置或场景对象之间的物理关系定义相机的位置。在实施应用之前，偏移值应从新局部坐标空间转换至世界坐标空间内。这里，偏移值使得相机可采用下列方式进行自身的定位操作，即玩家角色位于相机和世界空间内任一位置之间。

技术要点

此处将根据目标对象的相对位置以及某一独立位置（由成员变量 mWorldPosition 加以标识）创建参考帧。另外，可相对于新坐标空间内的目标对象使用相机偏移值。

```
Mat3 rotation = Mat3::LookAt(GetTargetObject()->
GetPosition().DropZ(), mWorldPosition.DropZ());
// note that the reference frame is often 2D, but is
```

```
// not required to be
mOffset = rotation * offset;
Vec3 desiredPosition = GetTargetObject()->GetPosition() + mOffset ;
```

局部偏移量

局部偏移量可采用与世界相对偏移量类似的方式获得，该操作不包括以下情形，即在应用程序使用之前，偏移量需要转换至相机提示对象或其他可控脚本对象的局部空间内。除此，无须其他对象确定当前坐标空间。需要注意的是，这并非是玩家角色的方向。另外，如果可控对象被重新定位，则局部偏移量也应进行相应的更新。

技术要点

这里，将根据脚本对象的方向创建新的参考帧。另外，可相对于新坐标空间内的目标对象使用相机偏移值。

```
Mat3 rotation = mScriptObject-> GetMatrix();
mOffset = rotation * offset;
Vec3 desiredPosition = GetTargetObject()->GetPosition() + mOffset ;
```

局部角偏移量

再次强调，该偏移量将以角度的方式加以定义，其方式类似于前述世界角偏移量。这里角偏移量将在可控脚本对象的局部空间或已定义的参考帧内进行定义。类似地，还应确定与目标对象之间的距离值。

技术要点

针对于偏移值，此处假设偏航角位于(0..2*pi)个弧度范围内，且围绕世界坐标系中的 Up 轴（此处为 Z 轴）呈顺时针旋转方向。另外，仰角则位于(0..2*pi/2)个弧度范围内，如下所示：

```
Vec3 offset(0.0f, cosf(pitch) * distance, sinf(pitch)* distance);
Mat3 rotation = mScriptObject-> GetMatrix();
mOffset = rotation * offset;
Vec3 desiredPosition = GetTargetObject()->GetPosition() + mOffset ;
```

相对于角色的偏移量

这里，期望位置将根据玩家角色人物的方向在局部空间内加以定义（或其他目标对象）。亦即，偏移量通过玩家的方向矩阵加以转换，进而获取真实场景中的位置。这对于第三人称相机而言可视为一类较为常见的解决方案，且常称作追逐相机（此时，相机可能在“追逐”目标对象）。多数情况下，根据围绕于世界 Up 轴的、角色前向向量的旋转操作，此类偏移量仅用于 2D 转换中（在平行于水平世界坐标轴的平面内）。此处，相机通常不会出现摇摆动作，同时，仰角也很少用于转换操作中。正如读者所预料的那样，基于角色的偏移可视为局部偏移的子集。

相对于角色的角偏移量

再次说明，该偏移量相对于玩家角色人物的方向，除非采用 3 个角度分量以及一个距离值。这里，鉴于角偏移量不可交换，因而其顺序显得十分重要。

当计算期望位置时，必要时，可向各轴向的角速度添加加速或减速效果，相应地，这向角运动提供了一定程度的延迟效果，该操作在实际应用中较为常见。首先，围绕角色垂直轴的角延迟增加了相应的美学效果，并使当前角色凸显三维特征；其次，延迟操作还可有效地防止角色行为过于“僵硬”。然而，该操作还可与视见位置结合使用（这里，视见位置不一定局限于角色之上）进而获得相对完美的视觉画面。

通常，相对于角色的偏移相机属性多与基于角色的角相机结合使用，因此，针对于围绕垂直轴的期望偏航角，垂直位置和距离（沿斜边）可描述为标量值。

如前所述，相对于角色的角偏移相机可视为局部角偏移相机的子集。

相对于对象的偏移量

由于此处并不考虑目标对象的方向，因而该偏移类型也称作非方向性相机。通常情况下，偏移量仅通过距目标对象以及距离值的高度值（角位移或垂直方向上的位移）加以定义。这里，目标对象与当前相机位置之间的向量定义了用于计算期望位置的坐标空间。多数时候，根据前述高度值定义的垂直分量，上述过程可视为一类 2D 计算。

基于对象的偏移量类似于基于角色的偏移量，且无须考虑目标对象的方向。当目标对象不包含明显的前向向量和/或可动态调整一段方向时，则是这一类偏移量的用武之地。当角色朝向相机时，偏移量的计算过程则无明显变化。这也使得相机可保持稳定状态且与角色的动态重定向和移动状态无关。

某些游戏则采用非方向性相机，但却允许玩家针对相机实施快速的重定位操作（位于角色后方），并通过操控按钮加以实现。同时，当按钮长时间按下时，相机的重定位操作将变得十分缓慢。

由于未采用距离和高度值且相机不存在隐式的重定位操作，因而当玩家操控相机并行进至某一期望位置时，对玩家各项属性进行多方考察将会是一种十分有效的问题处理手段，即使玩家从未介入，非方向型相机也会尾随当前移动角色，且不会受到角色动态重定向操作的影响。

针对于缺乏明显朝向方向的角色（如 Metroid Prime 系列游戏中的球体），当对该类型相机实施重定位操作时，角色的移动方向可用于确定前向方向。在计算过程中，为了有效地避免抖动噪声所产生的负面效果，当定义移动方向时，应使用相应的高通过滤器或阈值。当自动执行相机的重定位操作时，应对玩家行为企图加以显著地描述。

某些较为成功的 3D 平台游戏则使用了多种偏移类型并对原有游戏需求加以微调，如距离值以及相关动作行为。

技术要点

该偏移量类型类似于世界对象相对偏移量。这里，将根据相机与目标对象之间的相对位置创建新的参考帧。另外，可相对于新坐标空间内的目标对象使用相机偏移值。

由于相机位置用于确定参考帧，因而仅需要根据距离和高度定义偏移量即可。

```
Mat3 rotation = Mat3::LookAt(GetPosition().DropZ(),
GetTargetObject()- > GetPosition().DropZ(),mWorldPosition);
// N.B. only elevation (here pitch) and distance are
// required
Vec3 offset(0.0f, cosf(pitch) * distance, sinf(pitch)* distance) ;
mOffset = rotation * offset;
Vec3 desiredPosition = GetTargetObject()->GetPosition() + mOffset ;
```

路径

相机的期望位置可限定于游戏场景中的某一预置路径上，以躲避相关几何体或展示特定的游戏体验风格。其中，路径的形状可定义为圆形（源于游戏对象所定义的 3 次样条）、线性连接的路点系列（通过相关公式生成，如椭圆公式、抛物线公式）或者其他预定义形状。路径上的移动可通过多种方法加以操控，其中包括玩家位置的投影或线性时间值（亦即，与遍历场景的全部时间量所对应的位置）。

除此之外，映射函数可用于从某一数据域中取值（时间值或相对于某一参考帧的位置数据，如各类样条）并映射至路径中的某一位置。有些时候，可根据当前游戏状态动态生成路径，如两部相机间的切换操作（如第一人称相机和第三人称相机）。

需要注意的是，相机无须固定于路径之上。这里，路径仅用于表示相机的期望位置（如位于相机前方晃动的胡萝卜）。此时，相机很可能会采取捷径，因此应确保相机可有效地躲避相关的碰撞几何体。类似地，路径还可采用圆柱体空间加以表示，且相机距路径的最大移动范围为该圆柱体的半径。

通常，一系列对象间的连接（也称作控制点）可形成一幅有向图，其连接顺序可于随后根据时间或其他参数确定相机的位置数据。多数情况下，控制点之间仅包含单连接。此处很可能会产生分支路径，这将在相机位置的确定过程中增加相应的复杂度。针对于此，可通过相关搜索技术确定路径的选择方案，例如，基于一定美学条件的 AI 路径搜索，进而可实现正确的取景操作并使相机距目标角色之间的距离保持在适宜范围内。相机沿其路径的移动方式多采用直接控制加以实现，因此，路径的选择方案应参照时间值或目标对象的位置加以选取。

当相机在对应路径移动时，可根据相机的位置触发相关事件。

技术要点

一般地，当相机到达或经过某一控制点时，应对相关事件执行初始化操作，该角色方案常依据路径上的时间值（或长度值）加以制定，而非距控制点的物理近似值，其原因在于，路径曲线并非真实穿过控制点，具体内容可参考“样条”一节。当相机移动呈非线性状态时，该情形应引起读者的足够重视。例如，如果相机的位置根据时间值加以参数化，而距离长度值用于判断路径是否经过某一控制点。这里，相机在一次更新循环中可能会经过控制点，然而该情形很可能未被识别，

当时间映射致使相机位置经过控制点，并在基于前次更新操作的时间间隔内返回时，则很可能出现上述情形。针对该问题，另一种考察方式是，更新操作呈离散状态，而相机移动则（理论上）呈现为连续状态。这里，可针对时间值而非距离值求解各控制点处的路径问题，从而避免上述问题的出现。亦即，计算各历经控制点处的时间值（且无须考察路径上的移动方向）。因此，针对某一控制点，若当前时间值与某一（或多个）时间解相等，则对应角色可对此作出某种

反应。在实际操作中，该信息可在路径数据初始化过程中进行预计算。

通常，路径的形状保持固定，而路径的方向或位置则无须如此。另外，路径常受控于其他对象，因而也需要执行相应的平移以及旋转操作，进而对参考帧加以维护。较为常见的例子是包含旋转元素的猜谜类游戏，其中，相机的运动方式需与游戏中的某一元素保持一致，且无须考虑该元素在场景内的方向。同时，该方式还可进一步降低较为特殊的相机脚本机制的出现频率。

从本质上讲，当在场景环境中定义路径的位置时，均应对方向和位置的范围进行相应的考察。

路径类型

在基于路径的相机移动过程中，存在多种方式以表现曲线的形状，进而形成相机的运动路径。这里所描述的路径类型其差别并不明显，但均提供了各自的属性以标明其适用范围。出于简洁性考量，此处假设路径各分段中的路径类型全部相同。在同一路径中采用不同的曲率通常十分有效，尽管这将增加计算相机位置时的复杂度。另外，曲线类型还可修改路径的形状，具体内容将在第 10 章加以讨论，这里仅对其加以简要描述。

线性类型

由一系列线性连接控制点构成的路径可视为一类最为简单的路径，如直线路径。多个连接点可构成一条路径，连接起始路点和结尾路点即可形成一条封闭的路径。该方案可能会导致下列问题的出现，即相机在经过各路点时可能会产生运动不连续性这一现象。即使各个路段均由直线构成，相机在各路段内的位置也无须使用到线性计算。若相机在给定控制点未停止，运动的不连续性才会有效。

圆形/椭圆形/螺旋形路径

这 3 种简单的路径类型具有相似的基本条件，即皆需要通过中心位置和半径加以标识。此处，路径应具备基于非轴对齐方式的重定向能力，因而需要一个标准化向量用于表示平面的 Up 向量；同时，圆形路径在该平面内进行描述。一种简单的方法是使用脚本对象的方向数据对该运动平面进行定义。从数学角度上讲，与样条数据相比，数据圆形路径相对简单且占用较小的处理器开销（具体内容可参考“样条”一节）。通过定义长轴和短轴而非单一的半径值，圆形路径可方便地扩展至椭圆路径。多数时候，这一类路径可简单地限制在某一段圆弧之上，而非圆或椭圆。圆形路径可采用三次样条实施近似计算，尽管与二次曲线相比，该过程的曲率和半径计算稍显复杂。另外，可根据某些外部因素对半径进行简单的调整（如玩家位置）。根据某些外部因素（如与玩家的 Z 轴相匹配）改变相机的 Z 轴可生成圆柱体或锥体运动方式。上述内容体现了纺锤状相机所涉及的基本元素，具体内容将在“轴旋转/纺锤体”一节中加以讨论。

技术要点

二维圆形路径或椭圆形路径源自基本椭圆公式，如下所示：

```
X = a * cos(theta); // a is the semi-major axis
Y = b * sin(theta); // b is the semi-minor axis
```

其中，0 <= theta <= 2*pi 且假设椭圆中心位于原点位置处。

根据上述公式，可在一段时间内调整长轴、短轴以及 theta 值，进而生成各种椭圆路径。例如，若 a 与 b 相等，则对应路径呈现为圆形；随时间变化调整 theta 值，则相机将在圆弧上执行往复运动。

类似地，theta 值可根据玩家与参考帧之间的位置加以调整。例如玩家与移动弧中心位置间的向量与既定世界轴之间的角度（亦即，围绕纺锤体的玩家相对运动）。令 theta 值等于玩家角度与某一常量之和，则相机距玩家之间将保持固定的角位移。这里，可对椭圆路径实施旋转操作，进而可与场景内的任意方向加以匹配，且该路径不仅限于 XY 水平面。

样条

样条可视为曲线路径的一种，其属性和形状可通过设置一定数量的控制点（有时也称作节点或关键点）加以实现。控制点的相对位置定义了曲线的形状以及空间限制条件，尽管在某些情况下控制点的方向用于定义样条曲率（根据曲线类型，各控制点内还可能包含附加属性）。

样条的种类多种多样，针对控制点与曲率之间的作用方式，以及曲线是否经过控制点，各类样条均体现了不同的属性特征。当对各样条使用相同的控制点时，根据样条类型，其曲率可能截然不同。关于样条曲线及其计算方法，读者可参考第 10 章以获取更多的内容。

拉伸路径

在某些情况下，基于路径的相机运动无须从真实路径处获取位置坐标中的全部 3 个分量。某一坐标（或两个坐标）可直接从另一游戏对象处获取，或根据游戏对象与参考点之间的相对位置加以定义。例如，有些时候，根据玩家的垂直位置定义相机的垂直位置将十分有效，并将相机的位置限制在 2D 路径以避免与几何体发生碰撞。此时，样条曲线可视为一类平行于世界 Up 轴的拉伸表面。显然，这将生成任意方向的拉伸样条。类似地，该路径可用于确定相机的垂直位置，而非相机在水平面中的位置。当然，这其中还将包含诸多变化，仅当使用相对简单的运动约束条件时，该类型相机才可有效。需要说明的是，当前拉伸型路径适用于前述任何一种类型的路径形状。

路径位置确定方案

一旦路径的形状和位置定义完毕，则需要相关方法确定沿路径方向上的相机期望位置。这里，存在多种方式对此加以计算，具体实施方案将取决于路径移动过程中的上下文环境。相关示例如下所示：

- ☑ 非交互式环境。电影视频序列多采用时间值定义相机路径位置，其中，流逝的时间量与沿样条方向上的相机位置之间存在直接映射关系。通常情况下，相机将在某一时间段内完成全路径移动。这里，一类较为灵活的方案是以非线性方式定义位置数据；亦即，时间与位置之间的映射方式将根据设计者的特定意图加以控制（如采用二维样条曲线，具体内容可参考随后的“技术要点”）。
- ☑ 玩家角色相对于另一对象的位置。例如，玩家位置半径与独立游戏对象或位置之间的比较。其他比较方案还包括 Z 坐标之间的比较。
- ☑ 相对于既定路径的玩家位置。该路径不同于定义相机位置的路径。在多数情况下，由于可提供相对简单的参考帧，相对位置路径仅是一条简单的直线。然而，若路径可模拟玩家的移动方式，其用途则不言而喻，当玩家在相对封闭的空间内移动时尤是如此，例如隧道，尽管此

时相机将沿不同路径移动。玩家位置通常被转换为标准值（例如 0～1），进而体现沿其他“相对位置”路径时相机的相对位置。随后，该值可作为映射函数的输入数据（相关内容可参考随后的“技术要点”），并生成真实相机移动路径上的标准化位置数据。通常，映射函数可通过二维图加以描述，其中，输入值（标准化玩家位置数据）以及输出值（标准化相机移动路径位置）可分别采用两个轴加以显示。这里，图形状可呈现为隐式状态（基于相关公式，如抛物线公式），抑或使用分段 Hermite 曲线进行显式定义。最终，移动路径上的标准化位置将被转换回当前游戏场景中的坐标数据。

☑ 玩家与路径最近点之间的距离。当沿路径轨迹进行测算时，相对于路径某一端点，该距离可用于判断玩家位于路径的哪一侧。另外，该距离通常沿当前路径进行计算。考虑到存在多种求解方案，若路径具有较大的曲率（如 U 型曲线），则最近点计算可能会导致问题的出现。相机的初始位置通常可通过相机脚本系统进行显式定义，且常作为连接控制点顺序的有效参考（例如，起始点为首个链接控制点）。当相机在封闭场景中移动时（如隧道），该方案则颇为有效，其中，目标对象可能会超越相机或与其交叠。此处，在相机移动路径上，相机与目标对象最近点之间应保持某一预置距离。然而，尽管存在上述种种限制，相机依然需要保持平滑移动状态。目标对象的突然加速以及不连续运动不应导致路径上相机的不连续运动。

技术要点

在实际操作过程中，如何确定期望位置？若不考虑具体实现过程，基于路径的移动推出会展示诸多元素特征以及随之附带的问题。为了获取路径上的位置数据，应提供对应函数以对任意给定路段的路径加以定义。

通常，相关计算过程往往取决于路段内的标准化位置数据。然而，基于计算函数的输入数据无须生成对等的位置数据。

为了实现任意位置的计算，首先需要确定路径的全部长度。除了计算全长之外，还应对各控制点间的距离值加以存储，进而可对所需路段实现快速确认。

（1）样条。在多数情况下，可采用蛮力法计算样条的长度，特别是数据从游戏工具中导出时。当然，该方法可视为一类近似处理方案，如下所示：

```
float length(0.0f);
for (int i = 0; i < controlPoints.Size(); + +i)
{
    Vec3 pathPosition = EvaluateSegment(i, 0.0f);
    // start of i'th segment
    controlPoints[i].mLength = length;
    // save length at this control point
    for (int j = 1; j.<= kMaxSegmentSlices; + + j)
    // includes next control point
    {
        Vec3 newPosition = EvaluateSegment(i,j/kMaxSegmentSlices);
        Vec3 delta = newPosition – pathPosition;
        length += delta.Magnitude();
        // add the " length " of this slice
        pathPosition = newPosition;
    }
} // length now holds approximate total length
```

这里，如果各路段的长度值均采用数值方法加以计算，则可生成快速、准确的处理方案。针对于三次多项式弧长计算，[VanVerth04]中定义了一种较好的求解方案（采用 Gaussian 积分法），该方案在样条曲线的计算过程中十分常见。当路段弧长计算完毕后，可简单地遍历当前路径的全部路段，进而获取最终的路径长度值。

上述方法具有一定的计算开销，因而往往通过离线的方式或在数据加载过程中执行相关计算。如果控制点可在游戏运行过程中调整其位置数据，则对样条长度实施重计算时的处理器开销可通过多帧的方式进行分摊，尽管长度的调整可能会产生其他负面效应，例如可导致样条中当前相机位置失效（这将使相机呈现为不连续或其他未知运动状态）。

根据样条类型，曲率自身也会发生变化，从而引发当前相机位置失效。在上述各示例中，相机位置可在短时间内实施插值计算。进而可在一定程度上消除曲线变化，并降低相机期望位置处的碰撞几率。当向现有样条添加点时，存在相关的数学方法以实现这一过程，且不会对当前曲率产生任何影响，如[Farin02]。

若全部路径长度以及控制点长度已知（需要注意的是，控制不一定位于路径之上），则针对任意给定长度值，可分别计算路段索引及其内部的相对位置。第 10 章将对各路径类型的计算方法加以详细讨论，注意，特定距离处的路径位置计算通常十分重要。随后，可使用映射函数将时间值或其他数据转换为路径上的某一位置数据。

（2）映射函数。相机路径位置的描述往往取决于相机的应用环境。针对采用时间值作为决定因素的相机而言，该过程通常较为直观。时间值与样条长度之间的映射可视为一类相对简单的处理方案，其中，完成路径遍历的全部时间值可用于确定沿路径上的相机相对距离值。当然，该线性映射并不支持相机的方向反转以及急停等操作。在该情况下，相机移动速度通常保持恒定。另外，相机移动过程中不存在加速或减速状态，这将产生突发的启动和结束动作。通常情况下，该行为并不可取且可通过定义相应的加速或减速期以避免上述问题。针对大多数内置于游戏中的非交互式视频序列，这一类简单的方案并不需要过多的相机位置控制操作。

这里，可通过映射函数显式地定义时间与位置之间的关系。针对任意给定时间值，该方案提供了相应的机制用于定义路径上的位置数据。需要注意的是，计算后的位置无须呈现连续状态（并非是调节镜头）。同时，该函数自身可采用多种形式加以显示，通常定义为参数函数，如 sine()。Bump 和 Ease 函数应用较为广泛，其导数在开始点以及结束点处为 0，这反映了相机平滑的加速和减速状态。

除此之外，相对灵活的方案是采用二维分段样条曲线（通常为 Hermite 曲线），这一点与某些动画软件所采用的方式类似。用户可通过相关控制点（或关键点）定义当前曲线，并获取各控制点的内切与外切数据。需要说明的是，两个邻接控制点、第一个控制点的外切数据以及第二个控制点的内切数据即可构建三次多项式，如 Hermite 曲线。样条曲线具有一定的编辑功能以定义相机的移动行为，其中包括路径双方向上的移动，以及某一时间段内相关位置处的悬停状态。除此之外，作为一类显式映射函数，样条曲线的用途还涉及相机属性的方方面面，如方向、视域等内容。

（3）映射函数的更多内容。如前所述，当使用映射函数确定相机的期望位置时，时间值并非是唯一的输入类型。在实际操作过程中，流逝时间值常用于非交互式视频序列中。

场景环境通常会限制相机的定位操作，同时，这也是多数路径相机所面临的环境，因此，相机的定位方案多根据玩家角色人物的位置加以确定。当玩家自由地穿梭于场景环境中时，上述情形尤为明显。针对于此，应确保相机位置的一致性以及平滑移动特征。该问题的一类处理方案是，根据玩家角色与其“移动路径”之间的相对位置构建相机路径定位方案。即使玩家未被限制在当前路径之上，此处依然可计算路径与玩家之间的最近点位置。其中，“玩家路径”的形状无须与玩家移动方式相吻合，尽管该情形通常十分有效。在多数情况下，针对玩家相对位置的定义方法，采用直线作为参考帧通常更加简单有效。若不考虑玩家的路径形状且可根据玩家沿路径的相对位置确定其当前位置，则可于随后将该相对值作为相机位置映射函数的输入数据。因此，可通过玩家角色人物的位置数据调整相机的路径位置。当玩家在游戏场景中自由行进时，鉴于设计者具有对相机位置的绝对控制权，因而上述方案其功能十分强大。另外，还可对上述理论进行适当的扩展并根据相机的位置调整其相关属性。针对于视域，当使用上述方法且玩家到达游戏场景的特定区域时，可对玩家实施有效的放大操作，进而展示特定的游戏体验元素，如一扇门。因此，一切皆有可能。

（4）路径上的最近点。如前所述，确定路径与世界相对位置（如玩家角色的原点）之间的最近位置是十分必要的。针对线性路径，该操作较为直观。

基于路径的移动

这里，路径的定义方法多种多样，某些较为常见的方案将于稍后进行讲解。需要注意的是，如无特殊说明，当相机沿其路径移动时，将不会与周围环境发生碰撞。最终，固定路径相机可适当地降低处理器的计算开销，且玩家角色对相机位置具有完全控制权，鉴于相机位置可得到有效的保障，因而路径相机尤其适用于封闭空间或复杂环境。

当相机沿既定路径移动时，相机设计者可在任意时刻向玩家提供最佳画面效果。然而，上述设计方案可能会丧失某些灵活性，且根据路径移动的处理方式，画面有时会显得过于“僵硬”。

关于相机的移动方式，另一项需要考察的内容是相机的速度。这里，相机应以匀速运动，亦或是加速或减速运动？线性匀速易于实现，但却会产生某些负面效果，如相机突然启动，或即刻停止，针对观察者而言，这将使画面失去和谐感。若仅存在两个路点可用，当相机靠近路径端点时，可简单地向速度添加阻尼效果。当路点数量超过两个时，这将导致相机在各个路点处均呈现为缓慢停止状态——该效果并非是期望中的结果。这里，应谨慎地跟踪相机的速度数据，当相机接近目标位置而非路点时，可适当地添加加速或减速效果。从本质上讲，该技术仅适用于包含两个路点的路径。

另一种方案则将连接路点视为简化的 Bezier 曲线——此时，相关数据将呈现为平滑的渐变状态，而当前路径并不经过全部路点。

针对上述各类情况，有必要获取相关方法以确定相机在路径上的最佳位置。当然，这与具体的游戏因素紧密相关。例如，相机需要跟踪与玩家之间的相对位置，并确保相机不被玩家所超越（如在封闭的隧道环境内）。或者，玩家可朝向或背离相机移动，但相关限制条件可使相机与玩家移动实现良好的匹配。

与玩家坐标进行匹配

必要时，各轴可与玩家角色位置坐标加以关联，进而生成 2D 样条。该过程类似于表面相机，然而基于玩家的坐标锁定功能将使得最终结果呈现为轴对齐状态。

路径上的最近点

针对特定的世界空间，路径最近点计算可能会导致问题的产生。首先，根据路径的本质特征，存在多种对应的计算方案；其次，当选取“最近”位置时，其他方案是否可行？

在某些情况下，视线可视为一项重要的考察内容；另外，当前位置距世界空间内或路径上某位置之间的距离亦同样重要。若玩家沿真实路径行进，则相机需沿路径并背离最近点方向加以偏移；否则，插值计算不可避免。

约束表面

在少数情况下，环境复杂度不包含简单的路径，或者，相机移动受限于两个运动轴上，但为了与玩家位置相匹配，该相机需要沿第 3 个轴向移动。

在其他场合，若物理形状未与几何体实现良好匹配，则相机移动方式需要适应于复杂的、具有一定制作难度的场景环境。

在上述示例中，相机可能受限于某一隐式表面上，通常为当前玩家向该表面的某一投影位置处。

这里，表面自身可定义为简单的平面、球体、圆柱体或者某些参数表面。注意，此处并不定义相机的位置数据，针对于相机的移动，该表面可视为一类限制条件，进而防止其运功穿越当前平面（平

面可定义为任意方向）。

这里，针对其他位置确定方案，该表面均可视为一类附加约束条件加以使用。

另外，该表面的尺寸或方向可能固定不变，或者随具体的需求条件加以调整。类似地，表面的位置也可固定不变（根据场景几何体的具体需求条件），或者受其他游戏对象的支配，其方向处理过程也与此类似。

若复杂几何体处于移动或旋转状态，且相机表面仍需要与该几何体实现全程匹配时，该技术则十分有效。例如，某些复杂的猜谜类游戏，在游戏进行过程中，游戏画面可能会围绕空间内的一点旋转。

此时，表面相机需要根据当前谜题维持某一方向，这将涉及基于谜题自身的旋转操作。

针对于表面相机，可根据某一插值数据构建表面约束属性。例如，考察圆柱体表面，此处可根据圆柱体空间内的玩家 Z 坐标调整圆锥体半径，进而生成圆锥体。

综上所述，插值操作具有强大的计算功能。

相对于玩家半径，改变相机的 Z 偏移量则是另一个例子。注意，插值类型的可变性同样需要引起读者的重视。

此处，针对某一既定范围，较为常见的方法是在最小/最大值之间进行线性插值计算。通过设置合适的变量值，插值操作将具备一定的灵活性。

较好的方法是生成一个参数表面，并包含相应的既定关键数据（Maya 软件即采用了此类动画方式）。

其中，这将大大降低相机属性的完全非线性操控的操作难度；另外，这也使得复杂的相机移动操作以简单、可编辑的方式加以执行。

表面偏移

根据表面类型，存在多种方式用于确定偏移量，进而定义相机与玩家位置之间的相对位置。首先，相对于投影至当前表面的玩家位置，该偏移量可用于表面空间内，且最终位置将限定于表面边界内。其次，偏移量可用于表面空间内，但应相对于玩家位置并于随后投影至该表面之上。最后，偏移量可用于世界空间内，并于随后投影至该表面之上。

表面类型

针对于相机约束表面，其类型异常丰富。实际上，这一类表面无须通过程序生成，可简单地参照碰撞几何体或其他环境表现因素。可能的话，还可将目标对象位置投影至当前表面之上，并采用表面法线数据确定最终位置。

针对不同的相机约束类型，下列内容显示了表面类型的使用方案：

- ☑ 球体。球体约束相机易于实现且十分有效。然而，该方案也会面临某些问题。例如，当相机经过某些轴对齐位置时（如北极或南极位置），球体表面易受到万向节死锁以及其他偶然因素的影响。
- ☑ 平面。（任意）平面均可通过世界坐标位置以及平面法线加以定义。在实际操作过程中，可对相机穿越平面时的移动方式实施某种限制，如特定的宽度和高度值且位于脚本对象的世界中心位置（尽管该过程还会受到其他对象的影响）。法线方向则无特殊要求，此处，可使用法线数据并将目标角色位置投影至当前平面上。随后，该投影可用于定义相机的期望位置。同时，还可在平面（局部）空间或其他参考空间内向该位置添加一个偏移值。除此之外，源位置可在平面投影操作之前执行偏移操作（位于世界或对象空间内）。

- ☑ 圆柱体。作为一类表面约束条件。另一种相对简单的图元数据是圆柱体。其中，圆柱体可通过中心点、法线（圆柱体主轴）、高度值以及半径值加以定义。另外，圆柱体底面可定义为椭圆，因而需要创建相应的长轴和短轴，而非半径值。下面的样条圆柱体类型阐述了圆柱体半径的另一种调整方式。
- ☑ 圆锥体。这里，圆锥体表面可视为除去顶端的圆锥体，进而可避免圆锥体顶点附近处的单形问题。圆锥体可采用与圆柱体类似的方式进行处理，但圆锥体半径则需要根据其高度值、目标对象的 Z 坐标以及其他输入值进行插值计算。
- ☑ 拉伸样条平面。针对上述基础平面表面，可通过二维样条（如分段式 Hermite 曲线）以及基于该平面的拉伸表面对其实施扩展操作。

这一类表面将使几何体显得更为平滑。关于相机的定位方案，若可生成标准化输入值（可能根据目标对象与参考帧之间的相对位置），则全部工作将转换为计算 Hermite 曲线，进而获得平面表面上方的高度值。

- ☑ 样条圆柱体。类似地，可沿某一圆柱体环绕样条平面进而创建非一致性圆柱体表面。其中，针对 Hermiye 曲线（或其他映射函数）的输入值对应于围绕圆柱体轴的角偏移。而源于映射函数的输出值则可视为对应角偏移处的圆柱体半径。该表面类型将创建与纺锤体相机行为类似的相机定位方案，具体内容可参考“轴旋转/纺锤体”一节。

这也可视为一类相机形式，其中，围绕圆锥体轴旋转的相机将生成变化的半径。

约束空间图

类似于约束表面所采用的方式，此处可限制相机的位置数据，进而维护既定空间内的位置数据。这里，空间类型并不重要，但出于简单性考量，多面体形状通常作为首选方案。需要说明的是，期望位置的确定方案不受当前空间体影响，一旦相机向期望位置行进，即可采用相应的位置约束条件。例如，可简单地计算当前表面位置并将其视为相机最大（或最小）半径值，且相机无须固定于某一位置处——通过该方式，圆柱体约束条件可转化为空间体约束条件。

技术要点

一般地，若当前形状通过简单的公式或边界检测加以表示，则碰撞检测并不能保证相机位于空间体内。例如，若假设相机采用空间内一点加以表示，则圆柱体空间（特别是轴对空间体）仅需检测半径值并确定相应的轴长即可。

```
Vec3 direction = currentPosition –cylinder.GetPosition();
direction.SetZ(0.0f);
// this is for a vertical axis-aligned cylinder
// In other orientations we would use the local
// space of the cylinder
float radius = direction.Magnitude();
if (radius > kMaxRadius)
radius = kMaxRadius;
direction = direction.AsNormalized() * radius;
Vec3 newPosition = cylinder.GetPosition();
newPosition += direction; // i.e. a 2D offset really
newPosition.SetZ(currentPosition.GetZ());
// assumes vertical alignment
```

取景操作

该操作用以维护屏幕内目标角色的位置（以及尺寸）数据，且无须考虑角色在场景中的方向问题；另外，该操作还可同时对多个角色进行取景，并确保角色至少部分可见或位于显示设备中的特定区域内。

除此之外，还可对相对方向以及垂直位移进行定义；相机位置可根据电影技术加以确定（如特写镜头、脸部拍摄、中景镜头等）。针对上述内容，角色相对的方向也可进行适当的定义（如侧镜头）。与相机取景以及定位相关的电影技术规则已十分成熟，并为相机处理方案提供了丰富的内容。

对象的取景操作可通过多种方式加以实现。首先，对象的屏幕尺寸可通过在场景内移动相机得以维护。从本质上讲，该操作假设相机与目标对象之间存在相应的移动空间。其次，尺寸数据的维护方式可通过适当地调整视域范围并运用缩放效果加以实现。最后，场景的渲染视觉画面也可进行相应的调整，进而获取相关对象的所需尺寸。针对于后者，鉴于该操作将导致某些图像负面效果，因而较少使用。在实际应用过程中，相机移动常与视域变化结合使用。从美学角度上讲，无论取景操作是否完美，都应对结果做出最终的选择。在游戏中，有时需要以特定的尺寸对目标对象进行察看，而多数时候，相机在非交互式视频序列播放过程中需要执行重定位操作（如角色被击中后的反作用力效果）。此时，玩家角色有可能部分位于视锥体外部——通常，该情形无伤大雅。针对玩家而言，理解角色以及当前环境或其他游戏对象之间的空间关系至关重要。

在多玩家游戏中，还会出现其他取景需求条件，具体内容将在后续章节中加以讨论。

技术要点

为了保持对象的屏幕尺寸，首先应了解该尺寸的具体大小。这里，若假设相机可自由行进至任意位置，则可忽略相机的场景漫游功能，并将注意力集中于相机的距离和方向上。然而，此处仍需要对相机的期望位置作出某种限制。当维护对象的特定屏幕尺寸时，相对于目标对象的期望位置其确定方案将会限制某些方案的选择。

针对游戏对象的渲染尺寸，存在多种方案可供选择。一类方法是使用与当前相机相同的视图矩阵将该对象渲染至离屏缓冲区中。鉴于当前任务仅关注对象的轮廓信息而非具体细节内容，因而该渲染操作仅需要相关深度信息。随后，可对缓冲区进行分析，进而确定对象的渲染范围。需要注意的是，根据当前相机方向，该范围将随对象相对方向的变化而改变。实际上，由于上述变化将导致相机出现大范围的移动，因而这一类变化效果通常不予考虑。一类替代方案是针对世界空间内的目标对象使用固定尺寸，作为对象的最大尺寸范围，该尺寸可通过手工或离线计算方式进行选择。需要说明的是，这一类范围数据常对应于当前对象的碰撞体包围盒。

当对象的范围确定完毕后，可于随后计算相机距离值，进而满足特定的对象尺寸要求。在计算该距离值的过程中，仅考察某一轴向即可，通常为垂直轴。再次强调，使用限制条件将不可避免地减少相机位置的多样化特征。当逆置视图转换矩阵时，最终的相机位置计算结果也将呈现为逆置状态。

一类相对简单的方法是经根据相机与目标对象之间的距离定义期望尺寸。例如，当目标对象依据期望尺寸进行渲染时，由于仅需保持相机的相对位置，因而上述计算过程不再重要。若相机与游戏对象之间实现运动关联，则相机将保持同一相对尺寸，但其运动状态稍显不自然。

基于对象的取景操作

该位置定义方案不仅考察对象的屏幕尺寸，还包括与其他对象或目标位置之间的相对位置。通过该方式，两对象皆保持可见并位于屏幕尺寸范围内。根据目标对象的移动状态，这将使得相机呈现为动态移动状态。因此，若相机距目标对象足够远，这将大大降低相机的移动范围，因此多数取景操作仅取决于对象的方向变化。除此之外，视域变化也可保证正确的取景操作。视域的动态变化将导致玩

家失去方向感，并对渲染操作带来负面效果，例如，根据投影视图转换，垂直直线将呈现为弯曲状态。

在 2D 游戏中，相机与背景平面之间常保持固定距离，因而相关操作仅包含平行于该平面的相机重定位操作。据此，2D/3D 混合相机可简单地视为 3D 相机。

另外，除去保证对象完全位于视锥体渲染范围内，对象的取景操作还包含额外的限制条件以满足相应的美学要求。例如，如果两个对象间的相对尺寸差异较大其二者之间的关系易于理解，则较小对象一方具有优先选取权，无论如何，游戏体验元素应置于首要地位。这里，应确保玩家角色可见及其相对于其他对象以及交互环境的可见性。

潜在取景需求

相机的定向和定位方案多种多样，相关方案将根据游戏体验要求、对象属性特征（如相对尺寸）以及环境元素发生变化。下列内容列举了某些较为常见的方案：

- ☑ 对象的屏幕尺寸。
- ☑ 多个对象均需要位于屏幕内（对象整体位于屏幕内或对象特定位置位于屏幕内）。
- ☑ 对象间的距离。
- ☑ 与其他对象间的相对位置。
- ☑ 与屏幕间的相对位置（而非渲染尺寸）。
- ☑ 基于相机的方向（例如，保持特定的轮廓）。
- ☑ 具有瞄准功能以及一定尺寸的武器系统。

轴旋转/纺锤体

轴旋转相机具有较大的灵活性，但其描述过程相对复杂。这一类相机的基本功能包括围绕世界坐标轴（或纺锤体）进行旋转；沿平行于该轴的方向移动，进而形成相关位置处的圆柱体空间。另外，该轴通常平行于世界坐标系统中的 Up 轴，但这并非是必需条件。这里，相关限制条件还包括相机围绕纺锤体的旋转半径以及纺锤体上的距离。通常情况下，上述条件均相对于场景脚本对象的位置和方向而定。通过实时地移动纺锤体轴的位置和/或方向，可形成某些有趣的效果。若经过精心设计，基于上述条件下的相机期望位置移动方案将具有较大的灵活性。

当玩家围绕静态对象或在封闭的圆柱体空间内移动时，纺锤体相机通常十分有效。纺锤体相机的期望位置可根据玩家与纺锤体之间的相对位置加以确定。当从纺锤体至玩家位置处投射一条直线时（与纺锤体呈正确的角度），这将构成一个向量，针对相机的期望位置，该向量可视为一个参考帧。在大多数情况下，该期望位置与当前向量实现对齐，但对应半径值则较大。当玩家围绕静态对象移动时，这可视为一类期望行为。

然而，当给定参考帧后，期望位置的计算过程依然存在变数，例如：

- ☑ 相对于参考帧向量或轴原点的角偏移。
- ☑ 相对于当前玩家位置或轴原点的轴偏移。
- ☑ 相对于纺锤体轴或当前玩家半径的径向偏移。

除此之外，还可进一步调整相机的方向以满足大范围的游戏体验需求。

- ☑ 从当前相机位置处沿投影向量方向察看纺锤体轴。
- ☑ 沿经过相机位置处的投影向量背向察看纺锤体。
- ☑ 相对于目标对象，面向或背向纺锤体添加角偏移。

轴旋转相机的强大功能源于采用了玩家与参考帧之间的位置数据，进而可调整不同的相机属性。若通过某一插值范围（或某一常量）考察相机的各项属性，则可使用相关映射函数将玩家位置转换为该范围内的对应值。下面通过一个特定的实例对此加以解释。

此处，玩家位置相对于纺锤体轴的半径值作为当前映射函数的输入值。其中，映射函数定义了相机相对于玩家的垂直位移。针对于线性映射，当玩家靠近半径时，则可令相机向高处移动，反之亦然。

需要说明的是，映射函数不仅局限于线性方式，当采用分段 Hermite 样条将插值数据映射至相机属性时，该方案将极具灵活性。当前，这需要相关方法的支持，进而可对样条曲线实施编辑或定义操作。一类方案类似于某些 3D 动画软件所使用的处理模式，其中，时间值可作为插值数据，并于随后映射至旋转角度或平移值上。

一旦实现了映射函数的通用定义机制，即可据此使用各类插值数据，如下所示：

- ☑ 目标对象距纺锤体轴之间的距离值（即半径），该计算过程通常在局部坐标空间内完成。
- ☑ 玩家距纺锤体原点处的轴向距离（即沿纺锤体轴向的相对高度值）。
- ☑ 在纺锤体轴的局部空间内，作为角度值的玩家位置数据。该角度可视为逆时针或顺时针方向上的相对角度值，而相对角度值则位于 0° ～180° 之间或其他适宜的角度变化范围。
- ☑ 相对于外部参考帧的玩家位置数据，如纺锤体路径。
- ☑ 基于参考向量的玩家方向数据，或纺锤体的原始方向。

不难发现，对应变化范围从小到大不一而等。与灵活性对应的问题是复杂度，但该问题值得关注。单纺锤体相机可复制较为复杂的移动行为，并间接回馈玩家的动作行为。当玩家位于复杂的场景环境中时，如 Boss 战，这一类相机处理方案则十分有效。

常见位置问题

针对于各类既定定位方案，依然会出现某些不合时宜的情况，例如选取位置难以满足现有条件，或无法针对相机实施有效的定位。相关示例如下：

- ☑ 期望位置应与游戏对象保持一致（如玩家角色），否则，当相机与角色过于接近时，该角色将会穿越相机的近剪裁面。
- ☑ 针对于包含当前环境的几何体而言，若期望位置距该几何体过近，将导致外部区域视觉画面产生负面效果。
- ☑ 若期望位置距离目标对象过远，则玩家无法获得有效的游戏视觉画面。
- ☑ 几何体或游戏对象阻碍玩家与目标对象之间的有效视线。

上述现象可通过放宽约束条件或对约束条件进行适当的调整加以改善，如调整期望位置相对于目标对象的高度值。另外，如果满足相应条件（如与游戏对象未发生交叠），则前次相机位置也可作为有效的定位方案。第 8 章将对遮挡问题加以详细的讨论。

方向

本小节将针对相机方向的定义以及操控问题考察各类可能的处理方案。作为游戏相机的重要内容之一，定向方案将极大地影响到玩家的游戏体验。拙劣的设计方案将导致游戏失去可玩性，严重时，

甚至会惹恼玩家。这一点在诸多游戏评测中均有所指出。

就玩家满意度而言，期望方向至关重要，其中还包括相机的重定位方式。本章将阐述不同风格的定向方式以及期望方式的确定方案；随后，还将对真实的相机重定位移动进行讨论（此处为旋转操作）；最后，将对相机定向操作中所遇到的相关问题加以分析。

在第 2 章中曾经提到，存在 3 个分量可对场景相机的方向加以定义。其描述方式多种多样，本小节将采用下列规则对其加以定义：

- ☑ 偏航角。该操作使得相机围绕 Up 轴进行旋转，同时亦定义了相机的前视方向。
- ☑ 仰角。该操作使得相机围绕右向轴（或左向轴）进行旋转，相对于相机的前视方向，仰角将调整相机的垂直方向。
- ☑ 倾斜角。该操作使得相机围绕前视方向进行旋转。需要注意的是，倾斜操作在大多数游戏中较少出现，除了某些较为特殊的车辆模拟器，其中，相机可在一定程度上呈现为倾斜状态，以与其车辆保持一致的方向（相机的旋转速度通常小于车辆的旋转速度）。该效果有时亦出现于电影视频序列中以增强相机高速运动时所产生的效果。另外，倾斜角通常小于 90°，在其使用环境中，相机的运动方式通常被限制在某一固定路径上，或绑定在行为受限的行进车之上。针对于后者，相机常在倾斜方向上添加相应的阻尼效果，进而与车辆的移动以及倾斜状态保持一致。
- ☑ 视点位置。该位置定义为相机在世界空间内的焦点位置，亦即，相机方向实现自身对齐并直接指向该位置处。
- ☑ 视点偏移。这可视为定义视点位置的另一种方案。此处，视点位置于玩家角色处发生偏移并采用角色空间加以计算，即相对于角色的朝向方向（或该角色的移动方向）。
- ☑ 视见方向。该方向对应于相机的前视向量，即“视点”或相机的“镜头指向”方向。这里，视见方向等价于单位前视向量与转换矩阵相乘后的对应结果，该矩阵由偏航角、仰角以及倾斜角等属性构成。

技术要点

游戏相机的重定位操作通常会涉及向量以及线性代数等知识，并以此构建旋转矩阵、执行向量一致性测试等。[VanVerth04]等对此类技术进行了较为全面的阐述。

关于相机系统中较为常见的数学函数，读者可参考第 9 章以获取更多内容。对此，较为典型的例子是定义相关函数以构建对应矩阵，其中，平移和前视向量分别由游戏场景中的源位置和目标位置加以提供。连接两个位置数据的标准化向量形成了新矩阵的前视向量，而 Right 向量以及 Up 向量的构建过程皆源于该向量（此处，应确保 Up 向量与世界 Up 向量指向同一方向）。

这一类函数通常定义为 LookAt()函数，相对于另一对象或游戏场景位置数据，该函数可对游戏对象实施重定位操作（即保持对象间的可见性）。

方向表现方法

相机方向也称作前视方向，沿此方向，相机将面向当前场景世界。另外，该术语有时也称作镜头或视野方向。相机方向常与围绕该方向上的旋转操作结合使用，并定义相机的 Up 轴和 Right 轴。该旋转操作通常指倾斜旋转。在多数情况下，相机的 Right 轴与直接坐标系统中的水平坐标平面平行，并通

过镜像标准世界坐标后得到的视见内容消除相机的倾斜以及定向操作。

在本书中，正旋转值代表顺时针旋转方向；而负旋转值则代表逆时针旋转方向，且旋转轴背向当前对象（并在该方向上进行观察）。考虑到不可逆性，因而旋转操作的顺序则变得十分重要。这也意味着，若旋转操作的执行顺序不同，则最终结果也会截然不同。关于坐标系统以及转换矩阵的构建过程，读者可参考[VanVerth04]、[Eberly03]以及[Foley90]以获取详细的解释。

多种彼此等价的表现形式均可用于定义相机的旋转操作，但相关行为还应视具体任务而定。这里，相关内容包括欧拉角、转换矩阵、轴角以及四元数。各种表达方式均包含各自的优缺点，且可根据具体任务实现相互转换。例如，四元数适用于方向插值计算。

下面将逐项对此加以考察。

欧拉角

欧拉角从概念上讲则不难理解。欧拉角包含了围绕各旋转轴的旋转量且各自独立存储（通常位于世界坐标空间内，但也存在特殊情况）。通常情况下，这一类角度值以角度或弧度的方式加以存储，而其顺序对最终的旋转效果具有显著的影响。此处，旋转顺序应与其他游戏系统所定义的顺序相匹配。例如，分别采用 45° 仰角、45° 倾斜角以及 45° 偏航角执行旋转操作，但交换最后两步的旋转顺序后，对应的结果则截然不同。

另外，欧拉角可整合至围绕某一轴向的单一旋转操作中。若旋转轴共面，则像欧拉角实施额外的旋转操作将会导致问题的出现。当旋转矩阵的前向向量与世界 Up 向量对齐时尤是如此。该情形常会导致无法预料的旋转结果，即万向节死锁。关于万向节死锁，读者可参考[Hanson05]以获取更多内容。类似地，两个不同欧拉角旋转之间的插值计算也会遇到同样的问题。针对于此，四元数可较好地解决这一问题。

转换矩阵

另一种较为常见的方向表达方式是，将相机的方向数据存储为 3×3 矩阵或类似的转换矩阵；同时，除了旋转和缩放内容之外，还可向矩阵添加平移分量。此处，可连接旋转矩阵的乘法操作并将其应用于后续的转换操作中。再次强调，旋转操作的顺序至关重要且不遵守交换律。如果矩阵中包含平移分量，则应在旋转操作之前加以提取并于随后对其进行恢复。

矩阵格式也可发生变化，如行优先或列优先。旋转操作可用于局部坐标系统或世界坐标系统中，针对于前者，当执行旋转操作时，需从矩阵中移除平移内容。

然而，针对浮点格式的转换表现方式，后续的旋转矩阵应用将导致计算过程缺乏应有的稳定性，因而向量分量可能无法反映正确的角度值。为了避免这一问题，矩阵应采用标准正交基的方式加以定义。同时，每帧还应对当前相机矩阵执行有效性测试。

轴角

轴角对在世界坐标空间内定义了一个单位向量，以及围绕该轴的一个旋转操作。这里，旋转方向

（即围绕该向量的顺时针或逆时针方向）通常取决于游戏引擎所采用的坐标系统。另外，旋转数据多采用弧度加以存储。

四元数

当处理旋转操作之间的插值计算或围绕任意轴的重定向操作时，四元数提供了非常简便的方法。与之前的轴角对相比，针对方向以及该方向上的旋转操作，四元数提供了截然不同且相对复杂的方式。另外，四元数之间也可执行平滑的插值计算。

需要注意的是，在进行插值计算之前，应确保源向量和目标向量处于非平行状态（重合或背向），并于随后针对二者实施插值计算。

[Hanson05]对四元数进行了较为深入的探讨。[Shoemake85]首次对四元数在计算机图形学中的应用提出了自身的观点；另外，大多数数学专著对此也均有提及，如[VanVerth04]。

期望方向的确定方案

一旦相机的方向表现方案确定完毕，下面需要针对其期望位置加以考察。期望位置的定义很大程度上取决于游戏体验元素以及相应的美学需求。然而，期望位置的确定过程中依然存在某些普遍特征，针对第三人称相机尤是如此。第一人称相机通过玩家方向可实现期望方向的自动定义；当采用全视角模式时，还可通过玩家控制器直接对其加以定义。即使如此，依然存在某些情况可导致相机方向发生变化，如玩家跳跃时相机仰角的自动变化。针对相机方向的确定方案，下面将考察某些主要方法。

固定方向

该方式可视为一类较为直观的相机方向定义方案。此处，固定方向是指相机方向不会发生任何变化。该类型相机在游戏中较为常见，其中，背景环境多采用预渲染方式进行处理，进而可降低实时渲染所导致的处理器开销。尽管相机的方向保持固定，但渲染对象的移动和方向则无太多限制条件（即无须确保这一类对象一定位于视锥体内）。

早期的冒险类游戏以及某些以恐怖生存为主题游戏类型均可视为定向相机的实例，此处，定向相机不一定要在场景世界中保持静止状态，尽管静态相机更为常见。这一类相机通常会受到其他对象的支配，并保持自身行为的一致性以反映玩家角色的运动行为。例如，玩家角色位于迷宫内，则定向相机可对玩家视野进行有效的限定。

当玩家角色在封闭空间内快速移动时，定向相机可提供稳定的环境视觉画面，且玩家不会针对显著的相机方向变化而失去方向感。鉴于方向固定，因而玩家角色的迅速移动可能会导致短时期内视线的丢失。

高度恒定可视为定向相机的一个显著特点。此处，仅对相机仰角加以限制，例如，可通过游戏脚本机制或玩家控制操作。后者常为第一人称相机所采用，其中，固定高度值通常十分有效，当保持高、低对象的场景画面时，这可确保玩家自由状态下的瞄准操作。需要注意的是，必要时，相机依然可采

用偏航角旋转操作。

类似于动态固定位置相机所采用的方法，无论相机初始化时是否处于活动状态，动态定向相机可简单地使用同一位置和方向数据。因此，这提供了游戏的快照功能，当某些游戏事件导致玩家操控处于失效状态下时，该功能将十分有效。同时，该类型相机无须使用到插值计算以及任何逻辑更新操作，除非后续视域发生变化。

该类型相机的主要应用体现在以下方面：

- ☑ 基于玩家操控的远程相机。
- ☑ 回放相机的动态定位方式。
- ☑ 针对非交互式游戏视频序列的动态相机定位方案。例如，玩家被 Boss 捕捉到、被舞动并被抛掷一旁。另外，相机画面保持静止还可用于强调某些非交互式视频序列，进而避免跟踪玩家角色时相机所产生的快速重定位操作。

当然，仅对包含固定方向的相机而言，快照功能并非必需。某些插值计算也可采用类似的方式加以考察，例如，原始相机方向在保持一段时间后，方可对新方向进行插值计算。

跟踪目标对象或位置

游戏相机的主要用途之一便是对目标对象或位置实施跟踪。亦即，当前相机尝试保持方向对齐，进而，基于相机的视平面，特定对象或世界坐标可维持某一固定位置（或屏幕尺寸）。该位置（在世界空间中）也称作视见位置。在某些情况下，相机方向可能会锁定于目标对象处（或相当于目标对象的某一特定位置处），进而导致目标对象在视口或输出设备中呈现为同一相对位置。相机方向通常落后于视见位置方向。根据上述延迟效果，可通过相机定向过程中的时间值与理想方向实施有效的匹配。此处，理想方向可通过相机位置与视见位置之间的直线构成。

通过该方式，相机可抵达理想方向处，但相关过程往往取决于目标对象的移动速度以及相机的重定位速度。当然，相机可能最终并未到达期望位置处，该情形亦不足为奇。

第一人称相机以及第三人称相机均包含目标对象的跟踪功能，但后者更为常见。

如前所述，当相机在水平方向上跟踪目标对象时，应将相机垂直方向上的仰角限定在某一特定角度。当目标对象在移动过程中发生错误时，抑或在垂直方向上出现迅速变化时（位于封闭空间内），该技术将十分有效。

视见偏移相对于角色的偏移量

在第三人称相机中，主相机的交点位置通常位于近玩家角色处，当然，这也可以根据具体的有效体验需求进行调整。这里可采用距玩家位置处的、期望视见位置的物理偏移作为视见位置偏移量。由于玩家多关注于相关角色与当前场景间的交互方式，以及正常游戏体验过程中角色周围的场景画面，因而这大大简化了交点的表现手法。

第三人称相机多采用固定的视见偏移，因此，相对于玩家角色，视见位置将位于固定位置处。通常，该位置定义为玩家角色的原点位置处，抑或其垂直上方的某一位置处。当生成一致性的角色画面时，这将对游戏体验带来负面效应。在大多数第三人称相机中，相对于角色的移动，相机应具备一定

的“前瞻”功能，进而在当前角色以及其他游戏元素之间实施某种判断。另外，某些武器装备具有一定尺寸以及瞄准系统，因而，玩家武器的指向位置则变得十分重要，而非仅对该角色进行察看。

固定视见位置的难点在于，该位置在角色方向发生变化时，将会围绕该角色的垂直轴产生振荡现象，当此位置位于玩家角色前方时尤是如此。

即使采用方向延迟，针对角色的小规模重定向操作，其方向变化量仍难以令人满意。这里，可添加相应的弹性效果以及阻尼效果，并在一段时间内将视见位置从当前位置移至新位置处——通过该方式，上述负面效果可降至最低程度。另外，使用高通过滤器移除角色少量移动时所产生的噪声也可视为一类有效的方法。当与正常的方向延迟结合使用时，即可有效地消除视见位置所产生的振荡效果。

当相机对期望视见位置进行持续跟踪时，其 Right 向量应与水平坐标平面保持平行，以避免出现倾斜现象。

需要说明的是，视见位置无须直接绑定至当前玩家角色之上。在大多数情况下，可根据玩家角色的速度对偏移量进行缩放计算，以使玩家可对角色前方更远处进行察看。同样，偏移量（在角色空间内进行计算）也可根据具体的游戏体验环境进行相应的调整。然而，当玩家移动被中断时，根据玩家速度简单地对偏移量进行缩放将会导致快速的相机重定向操作，这将使玩家感到不知所措。而且，当玩家角色迅速改变其方向时，相机将对这一类变化实施精确的跟踪，并导致全部场景产生快速的移动或不规则的抖动现象。

为了有效地解决这一类问题，此处可根据玩家速度（以及其他条件）计算期望视见位置。通过使用比例调节器、PID 控制器或弹性效果向当前运动添加阻尼效果，当前视见位置可在一段时间内移至期望位置处。除此之外，在视见位置的移动过程中（或相机自身的期望位置）使用数字过滤器可有效地降低目标对象移动过程中对于非正常状态的敏感度。本章后续内容以及第 10 章还会对此类问题加以详细的分析。

锁定视点位置

第一人称相机可通过“锁定”机制确定其方向，亦即，玩家可在一段时间内被重定向，以在 2D 或 3D 环境下完成空间内的某一对象或一点的方向匹配。一旦方向匹配，则玩家角色可实现方向间的同步操作。由于第一人称相机与玩家采用了同一指向（至少为 2D 形式），因而在一段时期内，相机同样需要在垂直方向上被重定向，进而实现期望视见位置的指向匹配。在这种情况下，视见位置可定位于距目标对象底部某一偏移位置处，当发生 Boss 战时尤是如此，其中，相对于当前对象可能会存在多个锁定位置。

目标对象位置预测

针对于相机的正确定向，方案之一是对目标对象的移动实施预测。如果执行无误，玩家可对部分场景事先察看，以进一步了解玩家角色与场景之间的关系。

对未知对象实施准确的预测可视为一类常见任务，这在标准的 AI 文本中屡见不鲜，该方式在很大程度上取决于目标对象的运动特征（及其内部逻辑）。针对于相机而言，则无须对目标对象的未来位置进行精准的预测，此处，可根据玩家移动企图以确定其对相机方向与位置的影响方式。

例如，在第三人称相机中，常可对相机实施有效的定向操作，以使其朝向目标对象的前向方向。该操作可通过计算视见位置得以完成，且对应位置在角色移动方向上具有固定距离值和高度值。

然而，角色快速的重定向变化将导致视见位置产生明显的移动，这并非是期望中的结果。该效果可通过多种方式加以改善。首先，可在当前位置与期望位置之间且在视见位置的移动过程中添加阻尼效果。其次，还可根据玩家的速度调整角色位于前方的期望距离。玩家的移动速度越快，则其前视视见范围就越大（此处，至少应确保角色部分位于屏幕内）。

针对目标对象，通过将视见位置前移，目标对象相对于当前屏幕的渲染位置也会相应地发生变化。这里，相机前视与玩家角色可见性之间存在着某种折中方案。

对象的位置预测机制对于相机的移动和漫游确定方式同样重要。具体内容将在第 8 章和第 9 章加以讨论。

对象取景

根据当前行为上下文环境并以有效方式显示游戏场景画面可视为一类高难度、复杂的设计任务。即使如此，该任务依然是相机系统设计过程中较为基础的内容。在大多数游戏中，设计人员往往对相机取景方式缺乏足够的重视，然而，该方式定义了玩家在游戏场景中的视见信息量以及相应的观察位置。

在第三人称相机中，这对于玩家角色的前视机制同样重要。另外，对象取景操作还可保证场景内不同角色或位置处于可见状态；同时，玩家角色也应保持可见并可较好地理解两个对象之间的相互关系。

针对于此，较好的实例是格斗类游戏。其中，两位玩家（或更多）均希望在屏幕上察看到其对应的角色。不难发现，若角色之间的距离增加，相对于当前角色，则相机需要适当地回退和/或增加视域范围。频繁地改变 FOV 并不可取，这将拉伸当前场景画面并大大降低画面的真实感；同时，由于场景的渲染内容也会随之增加，该操作还会产生帧速率问题。

视见位置的确定方案

如果不借助于某些渲染技巧，计算角色的屏幕范围将变得十分困难。此处需要在屏幕空间内计算角色的边界范围，并于随后获取某一有效位置以确保全部角色均处于可见状态。当然，还存在其他“点对点”解决方案以降低上述方案所产生的性能开销。最为简单的方案是针对当前角色采用包围盒技术，并于随后将 8 个顶点坐标转换至屏幕空间内（如通过透视转换）。包围盒可近似模拟当前对象的轮廓形状，尽管该方式稍显粗糙。一类可控方案是使用定位器对对象相关部分加以标记，因而可对位于屏幕边界内的对象部分加以显示。通过此类方式，设计者可根据游戏体验需求条件动态地调整相机的焦点。同时，该方案适用于传统的“Boss 战”，当面对超出屏幕尺寸的大型敌方角色时，此情形常需要对当前玩家角色进行取景操作。

美学条件与游戏体验需求

电影视频技术多出现于对屏幕对象进行动态取景时。然而，该技术常会与业已成熟的游戏理论产

生冲突，该问题主要集中于角色的取景过程中，而非交互过程中的相机定位。当相机系统对动作序列取景时，环境限制条件表现得尤为强烈，当需要保证多个敌方角色均可见时尤是如此。电影技术具有表现相机移动行为以及角色定位的功效，进而加深观众对于演员表演行为的理解（或强调某种情感状态）；而处理对应的交互式电影视频时，其画面内容则相对简单。然而，当设计并实现相机系统时，对影视技术的深入理解将有助于提升开发人员的设计能力，其应用范围有时不仅局限于视频序列。在游戏体验过程中，某些特殊情况也可通过电影视频技术加以解决，针对实时、非交互式相机系统的电影视频技术，读者可参考第 4 章以获取更多的内容。

回放相机

由于回放相机隶属于非交互式范畴，因而相机可实现自动取景操作。通常情况下，回放相机将根据不同的视点位置对玩家行为动作加以展现，其选择方案应满足一定的美学条件。在某些游戏类型中，如竞技类或赛车类游戏，回放相机则是游戏过程中的重要组成部分。该类型游戏通常采用较为规范的场景环境（如场馆或赛道），鉴于设计人员可显式地对相机进行定位，因而这极大地简化了回放相机的操作过程。其他游戏也可借鉴该技术，尽管其角度变化幅度较大且偶尔情况下还需要玩家选取相应的角度值。为了创建动态变化的相机视点，设计人员应根据相机与目标对象之间的相对位置定义一系列的规则。

视域

一类常见的角色取景方案是调整相机的视域范围。在某种程度上，该操作方式与改变真实相机的焦距十分类似。然而，增加视域范围仅可实现有限的缩放效果，其原因在于，透视转换使得视觉画面产生变形。

无论如何，视域的微调并结合相机的重定位操作可在很大程度上辅助相机的取景操作。需要注意的是，应避免视域产生突然变化。在一段时间内，若与相机的移动拍摄结合使用，则视域插值计算将会十分有效。因此，画面中的角色将保持固定的屏幕尺寸，而背景图案则随着相机的移动方向变得大小不一。该效果常用于影视技术中（即移动缩放），进而强调角色对于事件的反应状态（如导演 Alfred Hitchcock 所执导的电影 Vertigo）。该技术虽然有效，但在正常的游戏投影过程中，这将会导致玩家失去方向感。

角色的空闲状态

当玩家角色处于空闲状态时，第一人称游戏（偶尔也会出现于第三人称游戏中）有时会采用半随机重定向操作，进而丰富画面的显示效果（例如，暂时终止玩家的控制输入）。该操作对应于玩家角色头部的轻微移动，当玩家在一段时期内处于静止状态时，这将向其添加某些细微的视觉效果；另外，还可降低由静态图像所导致的屏幕闪烁现象。这里，画面的移动应呈现为平滑的状态，且可通过正弦计算或基于控制点的样条路径加以实现。通常情况下，玩家在处于空闲状态时应确保处于屏幕范围内，

且在返回至游戏内部相机操控状态时其过程应尽量快捷、平滑。

自动方向操控

在某些情况下，相机的方向可通过游戏自身加以控制和调整，且无须玩家干预。当与游戏元素实现交互时，该模式无须玩家对相机进行任何控制，这在一定程度上对玩家起到了辅助效果。某些时候，重定向操作并不仅限于相机，玩家角色亦位列其中。相关示例包括以下内容：

玩家跳跃时相机仰角的自动控制

通常，该操作可在跳跃动作行结束时加以使用，进而改善着陆点处的视野画面（针对第一人称相机尤是如此）。此时，仰角下调且在玩家着陆点处逐渐恢复至水平状态，亦即，该角度调整为默认方向或静止时的方向。若玩家的跳跃范围已知，则相机角度的下调过程可从玩家角色上行时开始。同时，相机还暂时可处于等待状态，直至角色到达最高点处并于随后下调仰角。需要指出的是，重定向操作应以平滑的方式加以执行。

穿越场景环境时仰角的自动控制

当在坡道上行进时，该方式可用于显示坡道的上行或下行画面，并在玩家行进过程中提供较好的视觉内容。这一类技术分别适用于第一人称相机和第三人称相机，且仅改变相机的“静止”方向，而玩家可围绕当前方向实现全视角操作。

如果玩家在坡道上转向，则在角色方向与坡道斜面之间接近 90° 时，视觉画面将返至平行于水平面之方向。当玩家继续转动时，则仰角亦上下移动，直至玩家方向再次与坡道平行。这里，仰角值则根据玩家沿坡道上行或下行不一而同。除此之外，仰角的最大值也将根据玩家与坡道之间的相对位置发生变化，进而确保相机的平滑转换。

类似方案也适用于下列情形，例如，玩家接近洞穴或悬崖时，其中，相机需要自动调整其角度值以提供下方位置处的对应画面。当玩家之间靠近边缘位置处时，应相应地增加仰角值；相反，从边缘处退离时，则需要减少仰角值。

战斗或交互过程中仰角的自控操作

游戏体验过程中常包含下列情形（尤其是第一人称相机），即玩家需要以某种方式与位于视锥体外部的某一对象进行交互，且玩家无法对当前相机进行直接操控。玩家可方便地对此实施手工操作，但这常会分散玩家的注意力，因而在玩家执行其他任务时（例如，玩家处于战斗状态），该操作将变得难以控制。在某些情况下，可对相机的仰角进行自动调整（上调）且常与玩家重定向操作结合使用。另一种情形发生于玩家与游戏对象（如机械设备）直接交互时，此时，主游戏暂时挂起（以禁止其他对象的交互过程）。针对于此，游戏 Prime3[Metroid07]可视为一类较为典型的示例，其中，玩家角色（第一人称相机）与各类游戏对象进行交互，除了操控游戏相机的绝对方向之外，游戏引擎还将对这些游戏对象实施重定位操作。

玩家或相机与目标对象之间的自动重定向操作

该操作可视为由玩家或脚本机制所引发的单次效果；抑或，当角色被锁定至敌方角色之上时，该操作也可产生一类恒定效果。针对于前者，此类操作将确保玩家对游戏事件有所警觉；或者针对于需要操控的游戏对象，玩家可朝向特定的游戏对象。当玩家方向与水平视见方向同步时尤是如此。在后

一个例子中，需针对目标对象实施快速的重定向操作（尽管阻尼效果使得重定向的开始和结束阶段呈现为平滑状态），并于随后锁定至目标对象之上直至“解锁”为止。如前所述，当玩家无法通过控制方案实现盘旋等动作效果时，上述持续重定向方案则十分有效。

玩家角色朝向同一方向时相机的重定位和重定向操作

针对与第三人称相机而言，该操作通过由玩家控制命令加以引发。而对于第一人称相机，该操作多与全视角操控有关。相对于前者，相机将围绕玩家角色以圆弧的方式移动，直至定位于角色的后方位置某处。在移动过程中，相机将执行重定向操作并朝向视见位置。因此，在移动阶段的末期，相机方向将与玩家角色的移动方向实现匹配。当然相机在移动过程中还应考虑到其与环境场景特征之间的关系。这可通过降低移动半径值（当以圆弧方式移动时）或快速移动相机并抵达期望位置等方式加以实现。

第一人称相机与第三人称相机之间的转换

在游戏体验过程中，常会出现第一人称相机与第三人称相机之间的视点切换，实际操作过程常根据移动和方向自动执行。一般情况下，该操作可在一段时间内得以完成并应保持应有的平滑性。第三人称相机位置通常稍高于玩家角色位置，因而相机方向多从水平方向处向下仰动。在转换过程中，方向间的插值计算并不能确保玩家计算人物实现最佳取景方案。

在相机背向第一人称位置的行进过程中，应禁止玩家角色的渲染操作，直至相机行进至安全距离范围内，进而可消除插值计算所导致的负面效果。通常，为了缓解该问题所产生的影响，玩家角色可在一段时间内呈现为淡出效果（一旦超出了相机的近剪裁面）。

第三人称相机与第一人称相机之间的转换

在第三人称相机至第一人称相机的转换过程中，相机将会移至某一路径，因而相机的最终方向将与第一人称相机方向进行匹配。该操作通常需要采用基于大角度重定向操作的扫掠移动方式。这可通过以下方式得以避免，即使用跳接镜头切换；抑或从相机位置至角色位置对角色实施重定向操作，并使其指向当前方向。此时，当相机重定向并指向移动方向时，相机可简单地以直线方式移动，同时，玩家角色亦执行重定向操作并指向同一方向。如果玩家角色方向在转换过程中发生变化，则相机路径也需要以同一方式加以调整，针对相机方向，这将产生额外的旋转变化。因此，应确保玩家获取正确的方向感。

上述所选示例表明，方向控制不仅局限于玩家操控。需要注意的是，上述操作模式仅作为玩家的一种辅助手段。如果玩家在游戏体验过程中常纠结于相机的位置或方向操作，则该相机系统无疑十分失败。

重定向方案

当相机的期望方向确定完毕后，因存在对应方案并以平滑方式对相机实施重定向操作。当上述过

程在具体操作时往往会产生冲突（事实的确如此），其中，部分方案将与期望的全视角处理方案产生冲突；另外，其他问题还包括相机的重定向速度。一段时期内相机的方向变化常称作旋转操作。旋转操作的应用方式亦多种多样，且常取决于具体的对象表现风格。另外，也可采用多种方式表现同一旋转操作。例如，欧拉角、四元数以及转换矩阵均可表达类似的旋转操作。但相关操作内容依然持有不同的属性特征，因此其应用范围还应视具体情况而定。

旋转操作的应用

最为简单的方式是将旋转操作直接设置为某一方向，且相机随即产生变化。该方案多用于显式的相机定向控制操作，如电影视频序列。在大多数游戏体验环境中，玩家常希望在一段时间内以相对平滑的方式对相机的方向进行调整，针对于此，可在各相关步骤中使用较小的变化量。

下面将考察几种不同的旋转操作方案，且暂不考虑其表现风格。

均匀角速度

针对于相机的方向，常见的解决方案是使用固定的变化率，并通过角速度加以定义。除此之外，还可距上次更新所流逝的时间值按比例进行调整。虽然该方案易于实现，但在具体应用过程中，方向变化在开始和结束阶段常会产生不连续现象，进而使得场景画面缺乏应有的和谐。

加速和减速

作为匀速角速度的一类替代方案，可向相机的旋转速度中添加连续的加速或减速效果。这将有效地改善旋转操作开始和结束阶段时的不连续现象，当然，如果减速操作在一段时间内无法迅速地将速度降低至合理范围内，则最终结果将会超出原期望方向。该类加速方案可用于第一人称游戏中，这里，减速率应高于加速率，进而将“移动过量”问题降至最低点。

角速度阻尼效果

向相机的速度数据添加阻尼效果可使相机移动的开始阶段和结束阶段呈现为平滑的运动效果，类似地，阻尼效果也可作用于相机的重定向操作中。旋转开始阶段的突然启动以及结束阶段的戛然而止均会分散玩家的注意力。针对于此，阻尼方案可有效地处理这一类问题。除非相机的重定向操作采用了特定的时间量，否则，与简单地使用恒定角速度相比，阻尼方案可生成较好的移动效果。这里，方向变化速度可适当地减少当前方向与期望方向之间的差异。针对平滑移动以及阻尼功能的更为全面的分类方案，读者可参考第 10 章以获取更多内容。

全视角阻尼效果

全视角相机包含了另一种较为重要的相机重定向阻尼功能。其中，针对玩家所导致的控制器噪声

以及非正常输入数据，可采用碰撞效果或其他简单的功能抑制相机的重定向操作，但必要时依然可保持相机的平滑移动。

碰撞功能可作用于当前角度与期望角度之间的某一 delta 值之上，且较大的 delta 值代表了快速移动方式，随后，delta 值不断减小，直至碰撞效果对其产生作用为止。另一类限制相机方向加速的相关方案是，针对于当前方向与期望方向之间的角度值，可按比例添加阻尼功能。

降低突然转向的发生几率

当相机在其目标上方垂直移动时，将会到达某一奇点位置。此时，某些正常的数学计算将无法继续执行，且偏航角将变为倾斜角；同时，相机的重定向位置常令人始料不及。即使在到达垂直轴之前，相机亦有可能出现 180° 重定向这一现象，进而对经过相机上方或下方位置处的对象进行跟踪。这一突如其来的转向过程常会使玩家失去方向感，皆因快速的重定向操作其控制权并非掌握于玩家手中。这一类突发型转向操作可通过以下方式得以缓解，即在每一帧中，针对围绕相机前视向量的倾斜旋转操作，限制其旋转量。当路经奇点位置时，相机将会围绕新方向进行旋转。上述操作过程多出现于下列情形，即相机 Up 向量需要与世界 Up 向量保持同一方向。如果相机可采用“倒置”方式加以设置，则该情形不会出现奇点现象。多数飞行模拟类游戏可倒置相机的 Up 向量，并于随后向当前相机变换使用 delta 转换。此时，通常需要正交化该转换操作，进而避免浮点误差累计现象，以消除转换过程出现的不稳定状态（即非标准化操作）。

重定向延迟

多数情况下，相机方向不应锁定于目标对象之方向上。虽然相机重定向运动应保持为平滑状态（即维持良好的帧一致性特征），但这里依然建议向相机的重定向操作引入少许延迟（即重定向延迟或简称为延迟）。当添加延迟效果后，相机将呈现为更加自然的运动模式，但仍然应保证目标角色位于当前屏幕范围内。

方向延迟直接对应于屏幕空间内的玩家角色位置变化。尽管如此，许多游戏并不包含重定向延迟操作。此处，延迟操作包含两种效果。首先，相机将直视当前角色，甚至会阻挡玩家针对该角色的前方视线；其次，相机方向依赖于当前角色的运动状态。另外，任何突发型方向变化或非玩家操控型移动（如碰撞所导致的移动）均会对当前屏幕画面产生影响，并会放大玩家移动过程中所产生的细小震动，这将对画面的大部分内容产生负面影响。

为了降低或完全消除上述现象所导致的负面影响，应适当地“过滤”相关视见位置，以消除细微移动所产生的影响并在目标位置的移动过程中添加阻尼效果。

同时，当在当前方向与目标方向之间执行插值计算时，可限制每帧所允许的角旋转量并以此达到延迟效果。基于四元数的视见功能则会维护相机的 Up 向量，由于这将提供最为直接的旋转变化，因而该功能十分有效。除此之外，还可根据当前方向与期望方向之间的夹角通过比例方式向角速度添加阻尼效果。同时，延迟还可通过弹性或反馈控制器引入，具体内容可参考“弹性和 PID 控制器”一节。

偏移

第三人称相机往往会忽略视见位置与目标位置二者之间的关系。通常情况下，设计人员往往会将主要精力集中于玩家角色自身。通过直视角色（或于垂直上方某处），通常可获取较好的角色视野及其

动画效果（该效果很大程度上受益于动画生成器）。然而，玩家在场景中取向何处则往往被我们所忽略。如果未予以妥善解决，上述操作将会产生较为明显的问题，当游戏涉及瞄准操作或基于一定范围的场景交互操作时尤是如此。

第三人称相机的瞄准功能其实现过程较为棘手，其原因基于下列事实：由于视点未必与发射物的行进方向对应，因而玩家确定设计方向时通常会面对较为复杂的情况。如果不对发射物的数量加以限制，则发射物形成的轨迹可在一定程度上辅助玩家的瞄准操作；否则，需要将目标位置投影至屏幕空间并渲染某一瞄准标记，这在飞行模拟类游戏中十分常见。当然，其他形式的视觉元素也可辅助玩家的瞄准操作。例如，可通过 HUD 元素对地方角色进行高亮显示。若玩家可锁定目标对象（亦即，采用相对于对象的控制方案），则可在很大程度上消除自由瞄准操作所导致的问题（当然，射击延迟依然会导致某些问题的出现）。

平滑效果和阻尼效果

针对相机平滑的定向操作。根据当前位置和期望位置之间的夹角按比例限制相机的角速度可视为一类较为直观的解决方案。

下列内容显示了基于三角学的简单曲线处理方案。

```
Real32 const kDampingAngle(30.0f *
gkRadiansPerDegree);
Real32 angle = acosf(CVector3f::
Dot(currentOrientation, desiredOrientation));
Real32 const kDampingFactor = Cmath::Limit(angle /
kDampingAngle, 1.0f); // linearly proportional
Real32 angularLimit = angularSpeed * deltaTime *
kDampingFactor;
CQuaternion newOrientation = CQuaternion::
LookAt(currentOrientation, desiredOrientation,
angularLimit);
```

控制阻尼效果的另一种方法是添加碰撞功能或简化功能。这里，相关曲线函数可将位于 0～1 之间的数字以线性方式转换回原有数值范围。碰撞以及简化模式的相关示例将在第 11 章加以讨论，读者也可参考[Glassner02]以获取更多内容。

针对于第一人称相机的“全视角”功能以及任意时刻的相机重定向操作，该内容十分重要。需要强调的是，平滑的连续移动以及保持当前角色和谐的视见内容可视为一类互斥条件。

弹性和 PID 控制器

这里，可采用弹性机制或反馈控制器（如 PID 控制器）对相机的重定向操作加以控制。出于重定向考量，此处可以平滑的方式减少当前方向与期望方向之间的夹角。

需要说明的是，在以平滑方式到达目标角度的行进过程中，控制器不应在目标角度附近处出现振荡现象。如果未对阻尼机制加以严格限制，则可通过弹性或反馈控制器对此加以模拟。在实际操作过程中，采用控制器方式实现上述行为虽然相对耗时，但却值得。当然，偶尔情况下，小幅度的振荡效果亦未尝不可。通常，回放相机和手持相机的模拟过程则无须采用完美无缺的相机定向方案。即使如此，相机的振荡现象也应降至最低点——此处，可适当地采用过量操作模式（一次或两次）且无须围

绕内部对象实施重定向操作。

视见位置插值计算

一类不同的相机重定向操控方案是对相机的视见位置执行插值计算。当相机行为发生变化时，保持目标对象的视见内容通常十分重要，并可在源相机和目标相机之间进行插值计算得以完成，若插值过程为全部实现，最终相机似乎与其目标对象渐行渐远，甚至可导致玩家完全位于视锥体之外。考虑到插值计算结束时相机将会再次位于原始目标对象的中心位置处，因而该计算过程将会在一定程度上分散玩家的注意力。此处，建议根据期望视见位置直接（或间接）考察相机的方向，而非寻找当前期望相机方向。上述两种方案并无不同，在某种程度上亦确实如此。

然而，当发生下列情形时，情况则变得有所不同，即针对于最终方向的搜索过程，视见位置的插值计算应确保相机保持应有的视见画面，而无须考虑插值率或相机的行为变化。

自由视角

自由视角是一类常见术语，定义了基于玩家操控的相机方向控制方案。自由视角多出现于第一人称相机，且在第三人称相机中亦十分有效。针对于参考帧，该操作方式常可调整相机的原始方向。然而，在第一人称相机和第三人称相机中，其处理方式依然存在差别。

第一人称自由视角

在某些情况下，可对自由视角进行某种限制且仅调整垂直相机方向，并采用与常规移动操控类似的方式对当前玩家角色实施旋转操作。

全视角功能体现了某些潜在的玩家操控问题。根据游戏设计方案以及当前硬件配置条件，当玩家角色处于运动状态时，有时并不建议对相机的方向实施操控。大多数第一人称相机系统则允许在移动过程中使用全视角功能，进而全程支持某些玩家移动技术，如盘旋效果，该技术的具体内容将在第 10 章加以讨论。从本质上讲，该方法使得玩家可围绕任意空间位置以圆弧方式运动（常用于辅助瞄准操作或场景漫游功能）。

另外，方向控制可仅限于垂直移动方向上（水平方向则锁定于玩家角色方向），但在某些情况下，则需要对相机方向实施全程操控，且与玩家移动无关。

针对全视角相机定向方案，有必要对最大垂直角度加以限制。例如，当相机方向与世界 Up 轴平行时，这可有效地避免万向节死锁以及其他棘手问题的出现。类似地，在第一人称游戏中，相对于玩家角色的前视方向，依然可在相机的水平旋转操作中添加相应的约束条件。上述限制条件相当于对角色部分躯干的旋转操作加以限制，例如相对于下体方向的上体旋转。当然，某些玩家角色类型并不需要这一类约束条件，例如源于车辆驾驶舱的第一人称视觉画面。除此之外，如果采用全视角方式对玩家角色加以操作，则这一类限制条件亦稍显多余，如大多数第一人称游戏。

在第一人称相机中，参考帧通常为基于玩家角色躯体的前视 2D 方向。有时，基本方向（或静止方向）常可通过玩家、游戏体验需求条件以及覆写后的脚本内容进行调整。当玩家的水平方向及其视见内容同步时，全视角操纵则等同于玩家角色的旋转操作。然而，从当前角色方向中分离出视见方向亦是十分必要的。

这一分离步骤使得移动方向有别于水平相机方向。当采用这一类独立的运动模式，且全视角操作结束时，玩家视见方向应重定向至物理移动方向上，许多有经验的玩家对运动以及视见方向之间的不一致性均有着较为深刻的理解。

针对于车辆模拟程序，该情况则更为常见，尽管该方式多采用较为独特的相机定向操作。例如通过玩家控制命令而非全视角操作。当然，全视角模式并非不可用，如炮塔的重定向操作。一般情况下，当对炮塔或类似的设备进行操作时，玩家通常不会再对车辆进行控制，这在机动车辆操控类游戏中尤为明显。

第三人称自由视角

在第三人称相机中，全视角操控模式不会对方向变化作出任何限制。通常，相机的位置是可调的，如相机穿越球体表面时。其中，球体位于相机视见位置的中心处，抑或位于玩家角色垂直上方位置处。

在该方案中，相机移动应避免穿越几何体；同时，相机应围绕当前角色并在限定范围内执行垂直方向的旋转操作。这需要适当地降低相机运动的球体半径值直至相关障碍物皆被清除。相应地，应避免相机行进至错误位置并对玩家产生遮挡。

通常情况下，有必要采取玩家—角色相对控制方案（或某些非相机空间方案）以确保相机所产生的变化不会影响到玩家控制参考帧。

当前，针对第三人称相机操控，还存在其他可替代方案。某些游戏需要玩家对未来状态进行先期规划，进而使得相机可相对于当前玩家角色或在既定限制条件下进行制定，如预置路径。同时，玩家还应对相机执行正确的重定向操作。需要注意的是，这一类重定向操作应保证玩家角色位于相机视锥体内。这里，一类实现方案是围绕期望视见方向进行锥形重定向操作，一旦相机控制结束，该操作使得相机可重定向回期望方向（相关内容可参考后续章节）。

玩家移动

此处，还应对以下情形进行考察，即在操控相机的同时，玩家是否可以移动相关角色。尽管该情形并非不可实现（取决于控制器分配方案），但针对玩家而言，其具体操作过程并非易事。在某些情况下，一类游戏设计方案还会要求玩家呈现为静止状态，进而强调该玩家正在完成一类较为特殊的动作。

自由视角定向方案

若不考虑相机类型，全视角定向通常可采取两种方式，即自身中心定向以及非中心定向。

自身中心全视角方案

在该方案中，游戏杆或其他输入设备将导致相机执行期望中的定向操作，并根据输入设备的移动方式生成对应的移动。一般地，游戏杆数据范围将映射为基于角色方向的定向角度（或特定的“静止”方向）。除此之外，游戏杆数据值还可映射为相机的旋转速度，且该速度将作用于当前相机方向。在上述两种情形中，释放游戏杆将导致相机方向返回至原始“静态”方向。当采用前一类映射机制时，相关数据反映了游戏杆的真实位置；当采用后一类反射机制时，相机系统将会检测到未出现任何输入数据，进而在一段时间内相机会返回至原始（静止）方向。在相机返回至静止方向的过程中，其速度可大于全视角操作速度，但必要时，仍然需要添加阻尼效果图保证平滑的重定向操作。

非中心全视角定向

非中心全视角方案并不定义相机的绝对方向。相反，输入值映射为角速度或角加速度并与相机定向发生作用。当游戏杆或其他输入数据结束操作时，该模式通常不存在进一步的方向变化。因此，相机保持当前方向且不会返回至默认的静止状态，根据输入设备的灵敏性，当游戏杆返回至中间位置时，可能会产生某些冗余运动。

这一类意外的方向变化往往难以令玩家满意，这将给精确的瞄准操作带来种种困难。针对于此，存在多种方法可将其影响降至最低。一种方法是针对方向速度采用非线性输入数据映射，并向中间输入位置添加较大的阻尼效果。除此之外，还可针对相机方向的加速状态添加阻尼效果，但需要注意的是，应防止相机行动过于缓慢。

当相机自身无法返回至中心位置处时，根据当前游戏体验状态，则无须添加任何方向变化。例如，在第一人称游戏中，多数时候，相机将自动实现仰角的上扬或下调，或由设计人员加以操控。又如，当玩家沿楼梯上行，针对于静止方向，上述方案将有助于玩家实现仰角的上扬，反之亦然。该方案通常可通过标记表面或显式脚本机制加以实现。

无论采用何种方法，一旦玩家可实现全视角方案且操控设备释放后，视觉画面不再返回至中间位置，则应尊重玩家的方向选择方案，即使自动仰角修正操作具备诸多优点。

若玩家仰角未呈现任何变化，或在一段时间内仰角未被采用，则可适当地应用这一类仰角方案。通常，难点往往在于如何确定玩家的意图，当然，有时可通过当前游戏体验环境的上下文加以判断（如格斗场面）。但是，保证这一类相机自动操控方案的和谐性往往还会产生其他关联性问题。

常见方向问题

当采用相机重定向方案时，相机方向常会呈现令人意想不到的结果，如旋转抑或其他未知变化。下面将考察某些较为常见的问题。

万向节死锁

万向节死锁问题具有一定的隐蔽性且多出现于回转仪等设备上，这一类设备可用于维护参考帧且不考虑机动设备（如太空飞船）的旋转方式。当机动设备的旋转操作导致回转仪丢失参考帧时（常会返回无效的方向信息），则称该现象为万向节死锁。此处，万向节死锁则发生于相机的前视方向与世界 Up 轴对齐之时。当出现这一现象时，常规的相机旋转计算将不再有效并会导致较为明显的随机（即时）方向变化。该问题可通过多种方式加以解决。一种方案是禁止相机的前视方向与 Up 轴平行；另外，采用四元数构建相机的转换矩阵也是一类有效的解决方案。若相机方向需要与世界 Up 轴保持对齐，则应显式地控制相机围绕该轴的方向，进而避免视见计算所导致的随机方向问题。视见计算将重新构建相机的转换数据，但若保持正 Up 向量，则该转换操作可能会出现问题。

垂直转向

针对于万向节死锁，类似的情形也会发生于第三人称相机中，即垂直转向问题。例如，相机的前

视向量近似平行世界 Up 轴，二者之间夹角约在 10° 以内。另外，相机 Up 向量应包含正垂直分量。即使相机方向无法实现完美的世界 Up 轴对齐，但玩家角色的移动速度依然可快于相机（相机也可能呈现为静止状态），这将导致玩家位于相机的上方（或下方）。最终，相机将围绕世界 Up 轴进行快速的重定向操作。如果无法避免相机产生这一类极限方向，则应采取适当的措施以限制每帧或每次更新时的倾斜量（即 Up 和 Right 向量方向上的移动）。上述状况较为常见，如电影视频序列或第三人称相机返回至某一角落时。限制转向范围或围绕世界 Up 轴的旋转操作可视为一类较为健壮的处理方案。

另外，不建议采用快速的相机重定向方案，这将令玩家感到极度不适并失去应有的方向感。针对于此，可限制相机的重定向速度，较为明智的做法是使用相机跳接镜头。另一种方法是冻结相机的方向，直至玩家移至距相机 2D 位置足够远的某一位置。此后，依然需要对相机的重定向速度加以限制，以避免画面事物方向感。需要注意的是，当冻结相机方向时，玩家应可方便地移至某一不可见位置。

倾斜操作

在游戏设计中，如无特殊说明，倾斜操作（即围绕相机前视方向的旋转操作）通常被归类于电影视频序列技术中。倾斜效果并非是我们感知场景世界的自然方式，对于大多数人而言，该效果常会导致某种不适感。当采用倾斜操作时，尤其是倾斜角速度过大时，其结果将令人感到极度不适。然而，经适当限制后，倾斜效果并非如此不堪。

首先，飞行模拟器或类似应用程序多从驾驶舱视角对当前场景世界加以呈现，针对于此，倾斜效果仍然不可或缺，进而可重现机动设备的侧转效果。实际上，侧滚倾斜效果源于飞行术语。尽管如此，某些玩家依然认为该运动方式令人难以适应，尤其是飞行器的外部视觉画面。

其次，一连串的飞行动作中倾斜操作也必不可少。当模拟飞机或其他飞行工具/角色的运动方式时，常会涉及偏航角的变化。这也意味着，该角色应围绕其前视向量进行旋转，并以此模拟作用于飞行工具之上的受力状态。实际上，相机常故意滑向某一侧进而强调该行为状态。这里，相机的倾斜量常根据下列方式加以确定，即路径数据或根据路径曲率以及最大倾斜量等限制条件通过程序方式进行求解。考虑到相应的美学要求，因而倾斜量的量化过程并非易事，而在某些游戏中，还会通过倾斜操作故意令玩家丧失方向感。该状况应谨慎处理且出现频率不可过高，否则，玩家将会感到极度不适应。

如前所述，倾斜效果有时会用于电影视频序列中，进而描述或强调某种特定的飞行状态，抑或对某一场景进行取景以使玩家对此实现某种反馈。

方向噪声

取决于所采用的重定向方案，相机方向对玩家移动或席位变化所做出的反应均会导致其前向向量出现震动或抖动现象。若无法实现帧间的一致性，少量的方向变化亦会对玩家产生负面影响。采用高通过滤器移除不必要的方向变化可有效地减少或消除这一问题。

快速方向变化

当处理快速移动的玩家角色或即时方向变化时，可强迫相机执行快速的重定向操作。类似地，若

玩家角色可移至相机的上方或下方处，这也将导致相机的快速重定向操作（具体内容可参考“垂直转向”一节）。无论如何，快速的方向变化将使玩家处于混乱状态，并为理解角色与相机之间的关系增加了难度。当然，玩家有意识地操作所产生的方向变化则不在此例。例如，玩家从内部驾驶舱视野切换至机舱外部视野所导致的视口变化。这一类相机切换操作还是可取的，当新参考帧与前次参考帧相似时尤是如此。另外，当相机重定向操作无法通过相机切换完成时，问题则变得相对严重。此时，相机每帧只能通过较大的角度值调整其方向，因而这将导致渲染图像序列的不连续性。相对于相机的移动，这一类不连续现象则令人更加难以忍受。

玩家角色的视锥体剪裁

相机定向的主要目标是以适当的方式对当前游戏视见进行取景操作，并针对当前特定的游戏体验环境提供最佳视觉效果。不难发现，美学要求在取景方式上饰演了重要角色，但对于游戏内部相机而言，这并非是主要考察目标。需要注意的是，美学条件并不会受限于相关的电影技术规则，实际上，前者相对于后者更多的则是依赖关系。在第三人称相机系统中，玩家角色的取景操作应具备一定的“超前性”。然而，这并不意味着该角色一定要位于显示设备的中心位置。通常，位于玩家角色前上方的相关位置处应保持可见。即使角色并未配备具有一定尺寸的武器装备，保持场景环境先期可见依然对玩家大有帮助。另外，当相机背离当前角色调整其方向时，应保持该角色位于视锥体内。若玩家角色位于屏幕之外，玩家常会对此感到迷惑不解并由此产生控制参考帧问题。然而，随着相机方向远离玩家，玩家屏幕位置也会随着发生变化，即玩家通常会向相反方向移动。通过构建某一安全区域，抑或相对屏幕中心位置对玩家角色的角度加以限制，相机可锁定于该玩家角色的相对方向上。这将阻止当前角色离屏显示；然而，若玩家的速度变化幅度较大且相机接近于上述角度极限，此操作依然会产生负面效果。使用定时器增加相机的重定向速度可在一定程度上缓解这一类问题。在多数情况下，玩家角色的移动屏幕位置（特别是在机动类游戏中，如赛车游戏或飞行模拟器）多用于强调角色的三维特征并可向游戏场景添加有趣的视觉元素。

本章小结

本章讨论了游戏场景内相机定位以及定向方式的重要性。其中，存在多种方案可对此加以定义。相关方案的选取构成了相机设计的主要内容。再次强调，针对既定游戏体验环境，其处理方案多呈现为多元化特征。亦即，设计人员可对多种方案加以整合以呈现令人满意的视觉画面。这里，针对位置和方向这一类相机系统的关键内容，读者也可尝试不同的方法对其加以定义。

第 8 章　漫游和遮挡技术

第三人称相机多用于复杂、受限环境中。通常，相机可自由地穿梭于该环境中，且不应受到任何阻碍，相机与渲染几何体以及其他游戏对象之间不可彼此穿越。除此之外，相机应持续提供适宜的玩家角色或者其他目标对象的视觉画面，并通过动态变化的游戏体验元素加以展示。针对玩家所处环境，还会存在其他游戏体验需求，以定义某些特定的相机属性，例如保持既定距离或取景位置。即使在良好的环境下，对这一类包含潜在冲突的命令进行维护也将是一项艰巨的任务。

这里，预计算或预定义移动路径可在一定程度上对相机的移动起到辅助效果，但其视见画面通常具有一定的局限性（由相机设计人员所定义）。这里，建议相机可对当前游戏环境进行回馈，而非设计人员所预定义的有限子集。当游戏体验与相机定位之间需彼此匹配时，该方案将十分有效。在相机抵达目标位置过程中，其动态确定方案常称作漫游机制。

本章将从技术层面对相机系统加以分析，并以此揭示相机相对于复杂环境的漫游机制。因此，有必要对场景环境的构建方案实现进一步考察，进而消除相机漫游过程中的相关问题。

同时，本章还将探讨另一个相关话题，即遮挡问题。该问题涉及相机的定位与移动等相关内容，进而可避免目标对象被当前环境或其他游戏对象所遮挡。这里，针对遮挡问题，设计人员应尽量尝试各种方案并有效地避免问题的出现。

与遮挡问题相关的话题是视线的决策方案。尽管该方案常用于遮挡测试，但也可作为安全模式的测试条件。这里，安全模式可确保相机包含有效的属性，例如相机位于封闭的游戏场景中，或维护目标对象的可见性。一旦基于安全模式的测试条件被触发，则常需要采用动态行为动作对当前问题做出及时调整。本章后续内容将对此进行详细讨论。

将相机视为 AI 游戏对象

在多数情况下，相机漫游所产生的相关问题多与人工智能（即 AI）领域相关。如前所述，（至少）针对游戏场景内的定位以及定向方案，相机可视为一类 AI 角色。另外，与其他类型的 AI 角色相比，相机具有更大的运动自由度。多数时候，相机往往呈现为浮动状态，其运动方式不会局限于某一对象表面上，且不会受到其他物理模拟操作的影响。相对于真实相机设备，虚拟相机系统的优势在于具有更大的自由度。

然而，即使相机可采用与 AI 角色类似的方式定义其路径搜索方案，但依然存在某些与美学需求相关的限制条件，如目标对象的取景操作、目标对象的距离以及躲避渲染几何体。

AI 漫游包含诸多领域（如机器人学）其适用于相机操作。特别地，如果希望相机的移动与动态游戏事件遥相呼应，则通常会涉及相机移动决策的自治方案。本书所列举的参考文献中包含了多个相关示例，如[Borenstein90]。

考虑到环境特征的多样性，此处假设相机位于由碰撞表面构成的封闭环境内，例如多边形网格，并包含额外的独立游戏对象。这里，封闭环境是指场景的表现方式由邻接表面构成，这一类表面可对全部常规游戏对象加以限制（即使如此，某些时候仍需要将相机定位于外部场景中）。相对于渲染场景表面，碰撞表面可视为一类独立的实体（尽管二者不存在本质的区别），并包含了与表面材质相关的辅助信息。通常，这一类信息存储于各个表面或子表面中。另外，此处还假设游戏引擎内定义了相应的光线投射机制，并可返回碰撞点数据以及碰撞点处的材质信息。除此之外，针对相关投射光线，还需要对符合条件的材质类型进行过滤。需要说明的是，本书并不涉及碰撞数据的优化和整合方案，关于碰撞系统设计以及实现方案，读者可参考[VandenBergen04]和[Ericson05]以获取更多内容。同时，第 9 章也将对相机碰撞检测机制所涉及的主要方法进行描述。

与反馈相机相比，预测场景通常可取得良好的漫游效果，并对遮挡玩家角色的环境变化进行较好的预测。当然，上述操作均会产生不同程度的性能开销。

漫游技术

漫游技术的本质特征包含以下内容，即相机根据潜在的遮挡效果或移动路径获取相应的环境数据。游戏机场景环境的特点将对漫游技术所选择的方案产生极大的影响，如数据表现方式以及游戏引擎所提供的功能。下面将考察下列两种漫游技术，即动态漫游技术和预定义漫游技术。

动态漫游技术

动态漫游技术不依赖于与相机位置或移动相关的预定义信息。相反，该技术将对游戏场景进行分析进而在每帧中确定相机的定位方式。考虑到这将生成大量的计算开销，因而存在多种方案针对处理器开销加以分摊。

当采用动态方案时，需要重点考察玩家的位置和行为动作，该过程并非易事。这也意味着，该过程难以对全部情况进行较为彻底的测试，且针对于当前既定漫游选择方案，也可能无法消除某些特殊的问题用例。下面将考察某些较为常见的动态漫游方案。

光线投射

最为简单且常见的方式是光线投射机制，即穿越虚拟场景的直线的数学投影。但采用三角学时，光线投射机制可有效地判断直线是否与场景中的碰撞几何体或对象几何体产生相交。正如所料，这将生成大量的计算开销，当试图呈现复杂场景环境时尤是如此。当实现相机系统时，应对测试量加以限制，这一点尤为重要。基于光线投射的分摊机制切实可用，并可对降低处理器开销起到明显的效果，当然，这不可避免地限制了漫游时的信息量。然而，漫游机制针对这一类信息无须每帧执行更新操作，因而与碰撞以及其他实时数据相比，相关决策方案相对稳定。类似地，降低比较数据集的数量也是一类有效的方案（如空间划分方案或材质过滤机制）。

光线投射机制常可向相机系统提供丰富的关联信息，如碰撞预测信息以及玩家角色遮挡信息。除此之外，还可对光线投射机制加以整合，进而生成更为详细的遮挡效果。由于光线投射计算常会（根据定义）涉及 3D 直线的投影计算，因而当光线穿越某些狭小的开孔时，可能会遗漏相应的碰撞信息。

通常情况下，该问题可通过增加光线的投射数量及位置加以解决，进而使其覆盖更大的领域。需要注意的是，这将产生额外的性能开销。另外，也可采用适合于相机移动的独立碰撞网格，进而消除上述开孔。

另一种方案是采用光线投射延迟机制，亦即，光线投射的统计信息将在多次更新操作后加以收集。随后，可生成基于关系碰撞的概率图以降低每次更新操作时光线投射数量。这里，概率图可用于确定相机移动时的“影响力”。例如，根据光线相对于相机的光线投射位置，可推断出相机的横向或垂直移动方式，且若光线被阻挡，则无须执行相应的碰撞计算。此处，光线投射结果可通过各种方式对潜在相机为主加以描述。首先，针对于各光线投射，可根据光线投射的成功率定义相应的权值或影响力因子。因此，未到达目标的光线则导致相机背向移动。

另一类方法则从目标对象至相机逆向投射光线。这一类光线投射结果可采用类似方式加以使用。如果某一条光线被截断，则该光线将对相机的移动产生影响，其影响程度常根据碰撞点至光线投射圆点的近似值加以测算，抑或根据特定的光线方向简单地采用预定义权值因子。

光线投射的问题之一便是光线与当前场景环境（或其他游戏对象）是否会产生大量的计算开销。除此之外，通常还会使用到碰撞表面的属性信息。该信息可用于判断碰撞或光线相交是否真实有效。通常，若相关过滤器设置正确，游戏相机常可忽略场景环境中的大部分内容。

技术要点

常见的光线投射实现方案多围绕于相机的预置位置并实现为固定的源位置排列方式，且光线由此向目标对象投射。需要注意的是，此处应确保各排列位置不与场景几何体相交。多数时候，光线位置常以网格或其他对称形状进行排列。各条光线可赋予相应的影响力因子，其方向可垂直、水平或二者兼之。若光线投射无法抵达目标对象的期望位置，则该影响力值作用于期望相机位置之上。另外，影响力值甚至可根据与目标对象之间的相对位置加以计算。相关伪代码如下所示：

```
For all raycasts
If ray cast successful
    No influence applied
If ray cast fails
    Scale influence factor by distance of ray cast collision from target
    Add influence to desired camera position
End
```

待全部影响力值计算完毕后，相机将朝向包含影响力值的期望位置处移动。需要注意的是，若采用该方式，则影响力值在各轴上的计算方式也将有所不同。

体空间投影

有别于相对简单的光线投射，该技术将场景空间体由相机位置向目标位置进行投影。这里，空间体多采用凸状球体（有时也称作胶囊体）；出于简单性以及性能考量，空间体也可采用长方体或圆柱体加以表示。其中，可围绕空间体的周长采用若干光线投射模拟当前空间体，通过该方式，可有效地缓解计算性能问题。针对于其他碰撞决策方案，应从潜在碰撞集中剔除无关紧要的对象或几何体。轴对齐包围盒的边界检测或类似技术也可在前期用于剔除某些对象。空间投影常用于判断相机是否与当前封闭空间匹配，或者针对某一对象的相对位置加以判断（例如，剔除操作完毕后）。通过将移动空间体从当前位置投影至期望位置，该技术还可确保相机移动的有效性。

球体/圆柱体碰撞

出于碰撞考量，相机常视为球体对象，部分原因在于球体碰撞计算相对直观，且球体通常可沿碰撞表面移动。从相机角度而言，碰撞球体半径可稍大于视锥体近剪裁面距离，这将防止相机与渲染几何体产生相交。仅当对象的碰撞几何体或当前环境“大于”渲染几何体时，该方案方为有效。

需要注意的是，上述距离值还取决于水平视野。理想情况下，碰撞体应包含视锥体近剪裁面的全部 4 个顶点。

除此之外，相机碰撞体还可视为一类简单的、追至方向轴对齐的圆柱体。当相机远离水平表面时，如地板或天花板，该近似方案可提供与球体相同的功效，但却可有效地降低性能开销。

类似的限制条件也适用于近剪裁面距离，且垂直方向上的碰撞还需考虑到相机的高度，这将影响到几何体与近剪裁面之间的相交方式。

在第一人称相机中，鉴于相机完全包含于玩家角色的包围体中，因而无须使用碰撞体或其他测试机制（若相机移动被阻止且玩家仍可自由行进，则该情形将令人十分沮丧）。然而，确保玩家碰撞包围体尺寸大于近剪裁面距离依然是必要的。另外，考虑到该技术在第三人称相机的普遍性，这将有效地避免渲染操作所产生的副作用。另外，如果相关游戏元素（如 AI 对象）未与当前玩家进行碰撞检测，或渲染几何体超出碰撞体范围，则依然存在近剪裁面相交问题。

动态路径搜索

实时或动态路径搜索与游戏场景 AI 对象的漫游机制具有相似之处。然而，与 AI 路径搜索相比，相机的路径搜索过程将更为严格，其中，美学条件饰演了较为重要的角色。亦即，针对场景内移动时的空间体定义，相机的路径搜索机制以提供相应的预计算信息。这里，用于表述这一类空间体的数据结构也多种多样，但应可通过（有向）连接信息对体空间进行遍历。传统的搜索算法可用于确定高效的体空间路径，例如 A*算法以及 Dijkstra 算法（这一类算法在大多数 AI 或计算机专著中均有所体现），且均设定相应的权值用于描述特定路径的走向。然而，与 AI 对象相比，相机的走向方案则截然不同。

一旦路径生成完毕，相机便会沿此路径移动直至到达目标位置或被其他事件所打断。上述操作通常源于下列情形，即当前路径需要实施重计算，进而描述玩家移动以及环境变化等内容。通常情况下，路径搜索机制可分为两种层次。其中，高层次方案用于确定相机在空间体之间的全局移动，并结合底层方案实施游戏对象、碰撞几何体的相交检测计算。该方案与 AI 方案之间的差别在于，后者将沿对象表面处理较长的移动路径（尽管有时会出现飞行或跳跃动作），而相机则多位于近目标位置处。对于相机而言，路径信息多用于碰撞检测而非全程移动路径。较长的移动路径常通过预定义方式加以解决，且不需要任何类型的漫游逻辑内容。

生成动态路径

由于新增路径的形状和长度无法根据游戏场景中的内建预置信息加以确定，因而路径生成机制与路径搜索机制二者之间存在着较大的差异。相反，游戏场景信息往往根据当前环境加以动态确定，必要时可生成大多数路径。而且，该路径很可能随时间发生变化，尽管这将会导致意外的相机移动。动态路径常用于以下情形，即标准漫游机制无法确定适宜的移动方式。

另一类技术则根据玩家角色移动所生成的信息进行动态路径生成操作。在多数环境下，相机的移动路径可遵循于玩家移动时所生成的历史信息。当玩家穿越某一封闭空间时（如门廊），则相机可采用

相同的路径进行移动。然而，这一类路径所生成的玩家视见内容通常稍显异样。因此，围绕玩家移动顶点生成空间体是十分必要的，进而获取相机移动时的潜在限制条件（如圆柱状空间体）。在相机相对于玩家角色的移动过程中，这一类空间体可用于展现相应的视见内容。

可见性和渲染方案

随着硬件对渲染操作的强力支持，相机期望位置的确定方案也层出不穷。这里，首先假设相机的最终位置位于相对于目标对象某处，且期望位置与目标对象之间保持可见性。随后，相对于目标对象，可向距离值以及期望位置的高度值添加约束条件。这将围绕目标对象生成潜在位置的圆柱形带状区域。当然，还可对此加以进一步限制，例如相对于目标对象的固定角位移。

当给定上述约束条件后，则可得到潜在位置子集并可执行可见性测试。如果从这一类潜在位置处渲染当前场景，并使相机朝向当前目标对象，则将导致场景画面根据遮挡几何体或游戏对象显示当前目标对象。若对该视效画面进行渲染并使用较为独特的颜色绘制目标对象，则可通过分析屏幕渲染缓冲区的方式判断当前目标对象是否被遮挡。如果对象可见，则可将该相机渲染位置视为可见位置。其他硬件加速技术也可从各潜在位置处判断目标对象的遮挡效果，其中包括深度缓冲区。

图形硬件的并行特征使得相关计算开销可通过分摊机制加以解决。当然，该方案依然存在某些值得注意的地方。首先，渲染场景的构建过程应避开各潜在位置；其次，尽管基于硬件支持的渲染操作更加迅捷，但这依然需要付出相应的代价。其中，GPU 和 CPU 开销可通过以下方式得以缓解，即固定着色（如关闭光照效果）；调整视锥体并将当前场景画面限制在目标对象附近处（包含扩远剪裁面距离）以及降低渲染场景的分辨率。

一旦确定了相机的潜在位置，随后即可计算相关路径或其他移动方案，进而使得相机从当前位置移至目标位置。虽然该方案在位置处理过程中工作良好，但并未考虑当前位置与目标位置之间的几何体插入现象。对此，可添加其他约束条件以消除这一类问题。[Halper01]等对此类技术进行了较为全面的阐述。

碰撞体

碰撞体源于前述标准光线投射技术，并在此基础上稍作变化。这里，碰撞体的基本原理可描述为：光线投射位置常处于变化状态中。其中，各位置均位于某一小型碰撞空间体内（通常为球体）。这一类“位置”对象通常称作碰撞体，相关位置往往被限制在场景环境中。同时，碰撞体一般围绕当前相机并以特定模式加以排列，如圆形或半圆形。当相机在场景内移动时，碰撞体将获取相对于相机理想位置的期望位置数据。

另外，碰撞体的排列方式将会影响到相机与当前环境的交互方式。同时，各个碰撞体均会向相机的视见位置投射出一条光线，或者某些对于玩家角色的预置点。光线投射的结果常对相机移动过程中的期望位置实施某种偏移操作。碰撞体的影响作用取决于下列因素：

- ☑ 碰撞体距离相机的物理距离一般在相机的局部空间内加以计算，且应用方式亦大同小异。
- ☑ 各个碰撞体的预定义影响因素可根据组内特定碰撞体位置加以确定。
- ☑ 各碰撞体至光线投射与几何体碰撞点之间的距离。
- ☑ 玩家的加速度或速度，且根据期望行为该操作仅在某单一轴向上执行（如根据玩家速度所导致的高度变化）。

这里，可对上述因素进行适当的组合并作用于某些或全部碰撞体之上。通常情况下，可存在多组碰撞体，且每组碰撞体均会受到不同因素的影响。同时，各类影响方式也有所不同，通常可采用加权

平均值或质心计算加以处理。这里，读者可根据特定游戏中的环境类型进行相关试验；另外，上述碰撞体还可依据场景环境的行为类型采用不同的排列方式。其中，较为常见的排列方式是圆形或半圆形。当然，排列方式还可根据玩家状态变化加以调整，这里，速度可视为较为常见的可变元素。

碰撞体以及其他光线投射技术常会受到“噪声”的影响，光线投射与环境特征之间的碰撞过程常会产生这一类“噪声”。注意，该环境特征不应对相机的移动产生任何影响。针对于影响相机运动方式的某些结果，相关环境元素应对此具备一定的过滤功能。当涉及相机位置确定方案中的某一影响因素时，表面属性标记不失为一类有效的方案。除此之外，标记表面还可视为一类运动约束条件，抑或阻止相机到达某一位置。最后，碰撞体的排列方式以及影响程度还可根据环境属性（如封闭空间、垂直柱状体等）或有效体验模式（如匹配当前玩家角色变化的移动特征）进行适当的调整。

技术要点

鉴于碰撞体的排列方式变化较大，下列算法框架可用于确定相机期望位置的影响程度。

```
stl::vector<Vec3> influence;
// might be useful as a class
for (int i = 0; i < colliders.size(); ++i)
{
influence.push_back(colliders[i].offset * colliders[i].GetWeighting());
// the weighting depends on line of sight and/or
// other factors such as the relative collider
// position
}
// will need to get an average or similar
return GetCentroid(influence);
```

当确定平均影响力度时，碰撞体的排列方式可对对应的计算方案产生显著的影响。若碰撞体限定于某一平面之上，则质心方案通常工作良好。当最终的影响力度计算完毕后，该数据应在碰撞体偏移空间内加以使用。

预定义漫游技术

这里，可对相机潜在移动方式采用预计算方式或根据当前环境特征采取预定义辅助方式，以避免相机移动过程中所出现的诸多问题（相机设计人员可对此进行显式地定义，抑或执行自动预计算）。相机漫游机制的预计算过程可采用多种方式加以实现，但相关技术针对于相机位置的计算通常会产生内存开销或时间开销，尽管后者并非十分重要。预计算机制通常在运行期内产生较低的处理器消耗。与漫游技术相关的辅助机制通常可通过游戏脚本对象加以展现，这一类对象多通过游戏设计人员的相关意图加以设置，因此需根据游戏体验或环境需求加以精心设计。第 6 章曾针对相机的移动以及定位探讨了脚本机制及其对应应用方式，然而，与特定的相机行为控制方案相比，此类脚本机制多借鉴于其他游戏对象漫游过程中的辅助信息。

其中，信息类型的应用方式亦多种多样，进而可辅助相机的漫游操作并执行有效的碰撞检测计算。需要说明的是，上述处理方案常与 AI 或机器人技术具有某些共性，本书参考文献中亦列举了某些论文资源。

下列内容列举了某些较为常见的方案。

预定义路径

存储全部路径可视为一类最为常见的预计算方案。鉴于预定义路径可针对相机位置（以及方向）实施全程操控，因而某些游戏均采用了此类方案并忽略玩家的动作行为（或直接借鉴当前玩家的动作行为）。然而，预置路径移动方式可能稍显死板，并会导致玩家缺乏相应的控制感。相反，当对复杂环境下的移动方式进行处理时，对相机位置采取直接控制不失为一类有效的方法，特别是当位置确定方案根据玩家位置或动作行为产生各种变化时。关于基于路径的相机移动方式，其更为全面的叙述可参考第 7 章“路径”一节。

路径移动行为

虽然相机可通过固定路径限制其移动方式，但样条曲线亦存在多种方式对其移动加以限制。

（1）约束型路径相机

该类相机将根据预置路径在场景内实施精准移动。通常情况下，相机不应偏离其运动轨迹，除非该路径不再有效。这也是较为常见的路径相机应用类型之一，因此，相机位置可得到完全操控，进而可有效地避免相机与场景环境之间的配置或相交等问题。其中，非交互式电影视频相机（以及静态相机和伺服定位相机）多采用此方式。

（2）随行型路径相机

在大多数情况下，约束路径上的相机移动方式常过于机械且缺乏设计人员所要求的自然感，尽管该路径上的相机也会呈现出较为平滑的移动状态。即使如此，依然存在某些方法以对相机的移动进行约束。随行型路径相机即提供了对应的约束机制，当然，其实现过程稍有不同。其中，路径仅描述了期望位置——相机达到期望位置的方式可通过多种方法加以定义——亦即，相机并非完全限定于当前路径之上。该情形类似于在相机前方设置一“诱饵”。相机将会搜寻由当前路径数据做定义的位置，并以直线方式朝向该位置行进（需要注意的是，该技术可忽略相关定位方案）。除此之外，该规则还可用于控制相机移动期望路径的背向距离。例如，路径可呈现为管状形式（通常为圆柱状），尽管交叉部分不再为圆形（可能为椭圆形或矩形）。实际上，路径空间体的形状可包含于节（knot）中，并使得相机沿路径的移动方式产生各种变化（包括管状路径的半径）。另一种方法是采用参数曲线，其中，可将输入参数视为相机沿路径上的位置数据。上述方案可实现相机的局部操控，且无须包含多条路径。当相机背向跟随路径移动时，其约束方式不可过于机械，这很可能导致不和谐的运动效果。针对于此，可采用阻尼弹性或吸引/排斥方案以对距离约束提供更为和谐的反馈机制。

预定义空间体

这里，可对相机的移动方式加以限制，并将其限定于预定义空间体范围内。其中，空间体的定义方式亦不一而同，并由简单的凸体形状构成，如球体、圆柱体、椭球体或立方体。另外，如其他游戏对象，空间体还可源于更为复杂的碰撞网格。当然，形状越为复杂，对应的漫游机制其实现过程也就越加困难。这里，体空间内的期望位置可通过不同方式加以定义。不难发现，前述章节中的相关技术依然有效，例如，相对于玩家角色的定位方案并在体空间内辅以额外的限制条件。针对于大多数动态处理方案，该方法切实有效，因此，预定义体空间限制条件应在初始方案确定完毕后方可使用。一般地，固定尺寸的空间体可大大简化相关处理过程，但实际情况也会随不同的游戏体验需求而发生变化。例如玩家与参考帧之间的相对位置。与路径行为相同，直接限制相机位置将导致相机产生不连续的运动状态。因此，可借助于外力或阻尼弹性效果以确保相机位于当前体空间内。

吸引/排斥作用

高效的漫游辅助方案可通过吸引或排斥因子加以实现，其中，当前相机被特定的预定义位置、区域或其他游戏对象所吸引（或排斥）。通常，这一类位置常通过脚本对象加以显式定义，并由相机设计人员进行设置。同时，还可对当前场景环境执行离线分析，进而实现自动定位方案。在实际操作过程中，若缺乏相关规则对此加以控制，其实现过程将变得十分困难。

吸引功能或排斥功能皆可根据玩家或相机的接近程度（或方向）并向当前相机施加外力得以实现。需要注意的是，上述外力的实现过程不应隶属于物理模拟范畴（即使该系统确实存在）。相反，外力仅作用于期望位置并在真实的相机移动过程中充当一种约束条件。

针对外力计算中的接近程度，吸引或排斥因子应呈现为球状分布，但这在具体应用中并不具备实际意义。在大多数情况下，圆柱体则更为常见，其主轴常与世界 Up 轴实现对齐。当对游戏对象之间的接近程度加以计算时，考虑到其计算开销，通常会选取简单形状作为首选方案。同时，这对于外力计算同样不可或缺。此处，常见方案包括球体、圆柱体以及平面。

一类较为特殊的情形是，相机在门廊中漫游，该过程易出现问题，当玩家快速移动时尤是如此。门应在相机与目标对象（或上方位置）之间保持开启状态，除此之外，其问题的出现还源于以下情形，即门廊多实现为某一封闭空间，且玩家角色存在多种移动方案——这将导致相机位置后滞。垂直门廊则更为常见，且门通常会与游戏场景的 Up 轴实现对齐。水平门廊则更具挑战性，尤其是相机方向可能会包含潜在的万向节死锁现象。其他问题还包括，外部事件导致门的关闭过程过于迅速，从而使得相机相对于已关闭的门对象处于滞后位置（更为严重的是，相机有可能与门对象几何体相交）。为了有效地解决这一类问题，当相机与玩家分置两侧时，门对象应保持开启状态。当然，该方法并非万无一失。针对于此，相关检测机制则变得十分必要，进而避免相机的位置产生滞后现象。当门关闭后，将使得相机位于正确一侧。

若相机系统可对相机与门廊之间的接近程度加以跟踪记录，抑或对相机与门廊之间的相对位置以及方向加以提示（如通过触发体），则可采用基于轴的吸引因子对相机的移动方式加以导向。根据相机与期望移动轴向之间的相对位置，可通过比例吸引因子向当前相机施加外力。需要注意的是，应确保外力作用不会产生震动现象；同时，还需对其应用场合进行仔细甄别。

针对此例，吸引因子或排斥因子所产生的作用力方向亦不一而同。通常，作用力常与吸引因子或排斥因子的物理分布形状相关，例如，球体吸引因子通常会将对象拉向其球心位置。另外，该过程还可通过世界相对方向加以实现，例如，圆柱状排斥因子可根据玩家角色与圆柱体中心轴之间的接近程度确定受力状态，进而基于固定世界方向施加排斥力。

有时，吸引因子和排斥还可同时绑定至其他游戏对象之上（或构成游戏对象中的一部分内容）。因此，其位置和作用效果均处于变化状态中。

当采用外力方案时，需要注意的是，应确保相关对象在必要时接受作用力，[Stout04]对此进行了较为详细的阐述。在实际应用过程中，应确保相机不受到反作用力的影响，这也是外力方案的难点所在。针对于此，相机通常有别于 AI 对象且需要满足特定的美学需求（例如，应确保相机处于平滑运动状态）。因此，如果相机偶然途径吸引因子所处区域，则不应受到任何影响。类似地，若相机背离吸引因子区域移动，则不应受到任何力作用。

综上所述，通过对吸引因子的位置加以精心布置，相机运动可有效地避免几何体碰撞问题，且在封闭场景中行进时无须采用相对复杂的、基于手工脚本的相机方案。实际上，吸引因子和排斥因子常

可与游戏对象结合使用（如门对象），进而使相机的移动方式摆脱脚本机制的束缚。

流场

流场可视为一类稀疏向量数组（具有方向特征），并采用跨越 2D 平面或 3D 空间的网格加以划分。其中，向量之间的间距无须实现均一划分。当相机（或其他对象）途径流场时，相机的移动方式将会受到其附近处向量的影响，且作用力方向为对应向量方向。同时，受力强度取决于相机与当前平面或其他影响因素之间的距离。这里，向量数组类似于对象周围流动的水流或其他类似流体，这也是流场这一名称的由来。通过调整向量的方向，可确保相机以统一方式避开当前障碍物。因此，应确保向量方向不会出现下列情形，即作用向量之间彼此产生抵消，从而导致相机呈现为静止状态，或使相机无法正常地避开当前障碍物。针对于相机的移动状态，流场可视为一类有效的辅助手段，但绝非是唯一因素。另外，有必要对这一类辅助作用的应用时机加以判断。例如，向量可能呈现为单向或双向，相对于流场向量方向，如果相机以不同的方向遍历流场，则仅当相机移动平行于该向量方向时，方可施加作用力。

除此之外，作用力的大小还将正比于相机的移动速度。

影响力地图

影响力地图则可视为另一类稀疏数组且在某种程度上与流场类似。然而，相机位置与影响力地图之间的映射方式多参照目标对象与影响力地图之间的相对位置，而非相机自身。影响力地图中的各个位置对应于相机所占据的空间位置。源于影响力地图中的期望位置常可用于插值数据，进而使得相机从当前位置处即呈现为平滑的运动状态。在某些方面，样条运动方式可视为一影响力地图一维变化版本。其中，相对于样条的目标对象位置可逆向映射为相机在该样条（或不同样条）上的某一位置。根据碰撞或视见信息，影响力地图可显式地加以定位或采用离线方式加以生成。

势场

势场源于机器人导航技术，该技术针对动态相机漫游以及碰撞检测不失为一类有效的解决方案。作为一类高级话题，其深入分析则超出了本书的讨论范围，下面仅对此加以简要的介绍。术语势场源于静电力原理，亦即，对象的受力效果与该对象和施力者之间的距离呈现为某种比例关系。另外，这一类作用力通常仅作用于处于活动状态的相机之上，以避免相机与渲染几何体之间发生碰撞现象。除此之外，相关作用力分量常施加于相机的运动轨迹之上，当相机在封闭空间内移动时，这将起到较好的辅助作用。通常，当相机与几何体之间趋于碰撞时，这将避免诸多问题的出现。另外，作用力还可作为点或平面表面加以显式定义，并由设计人员置于当前环境内，该方式与吸引和排斥因子类似。需要说明的是，作为场景数据的部分内容，作用力的位置和方向亦可自动生成。

类似地，碰撞几何体也可用于生成相关作用力。然而，碰撞几何体的复杂特征将使得相机最近处的受力方案产生较大的计算开销。一类可能的处理方案是生成隐式表面，该隐式表面可以一种平滑、连续的方式靠近当前碰撞几何体，进而实现距相机最近位置处的快速定位方案。关于隐式表面，读者可参考[Bloomenthal97]以获取更多内容。

同时，机器人技术也对上述各项方案有所涉及（参见[Koren91]），关于场景环境的复杂度，亦存在多种相关限制条件，针对封闭空间则尤为明显。需要注意的是，同步逆向受力方式还会导致振荡问题的出现（如在狭窄的走廊中）。关于游戏势场的具体应用方案，读者可参考[Stout04]以获取更多内容。[Borenstein90]中也提供了一种有效的替代方案，即向量场直方图（VFH）。简而言之，VFH 存储了障碍物与当前对象之间的距离信息（通过红外线传感器或其他类型的传感器加以确定）。这一类传感器通常架设于固定角偏移处，或围绕某一中心点位置稍作旋转，并向当前场景环境中投射光线。这里，可在某一时间段内对每个角偏移处所返回的距离信息进行分析。随后，即可创建已知对象的直方图，并可向潜在的碰撞处理方案提供足够的信息。

环境遮挡

本小节将探讨玩家角色的遮挡问题及其对当前游戏体验所产生的影响。根据角色和相机之间的环境特征或其他游戏特征，当玩家与角色之间的视线被阻挡，则会产生遮挡现象。根据定义，由于相机移动无法很好地处理二维环境下玩家角色的遮挡问题，因而该问题更多地出现于第三人称相机中。

在第三人称相机中，遮挡问题通常比较棘手，且需要根据游戏类型、当前场景以及游戏体验环境并结合多种方案加以处理。在某些场合中，甚至会采用跳转镜头将相机移至更加合理的位置或方向处。针对相机的位置和方向，本小节将集中探讨如何避免产生遮挡问题（以及何时该问题处于可接受范围），并对其具体的解决方案以及预测方案提出相应的建议方案。

另外，环境遮挡测试多取决于当前碰撞系统所提供的相关功能项。稍后将会发现，较为典型的环境遮挡测试方案通常需要得到光线投射机制以及游戏场景中的空间体的支持，进而对视线或相机新位置处的可见性进行测试。

如何看待环境遮挡问题

环境遮挡问题可采用多种方式加以定义，且多取决于当前玩家角色的本质特征。这里，环境遮挡定义为：玩家无法对其角色实施有效的观察，且二者之间的位置、移动方式以及角色与其他游戏体验元素之间的交互操作缺乏应有的控制环节。该情形常与环境几何体以及位于相机和角色间的其他游戏对象紧密相关。基于玩家操控的角色可见性常取决于该角色与相机之间的距离以及玩家角色的物理特征（如尺寸和轮廓信息）。当对场景环境中玩家角色的位置和方向进行操控且需要保持应有的运动和谐性时（如角色可能或从某一高处下落），环境遮挡问题将变得十分重要。

针对于玩家角色如何获取相对流畅的无遮挡视觉画面，该问题在第三人称相机环境下常未引起足够重视，这可视为一类常识性的错误。主玩家角色的遮挡现象迫使当前角色对相机和/或玩家角色实施有效的操控，进而获取相应的视见内容并满足一定的可控性。在某些情况下，该任务则难以实现，这将大大降低玩家的游戏体验感。鉴于某些游戏并不支持全方位的相机操控，或对此有明显的抵触现象（根据相关设计要求或其他人为因素），主角色的遮挡问题将对游戏体验产生直接的影响，这在场景漫游时表现得尤为明显。除此之外，若玩家对相机采取直接控制方式，针对其角色执行其他动作行为将变得十分困难，当游戏支持多按键操作或多设备操作时尤是如此。更为糟糕的是，某些操控环境还会采用特定模式（如对话框）对相机的位置或方向加以控制，这将使玩家与游戏之间的交互操作变得极

为不和谐。

当然，可采用相应的折中方案对此加以处理，尽管如此，仍需要对具体操作过程实施多方考察。多数时候，角色的部分遮挡尚可接受，例如，角色尺寸超出屏幕范围或遮挡区域相对较小。相反，完全遮挡则不足以取。角色距离相机越远，则获得当前角色的无遮挡视觉画面就变得越发重要。需要注意的是，即使全遮挡效果允许存在，其出现时间通常也会十分短暂。实际上，短时期内的全遮挡效果可有效地增强玩家的临场感。其中，这一类效果主要用于强调当前玩家角色隶属于场景中的某一部分内容，因而允许出现较为短暂的遮挡效果。例如，玩家从垂直柱状对象后方经过并迅速出现于另一侧。当然，若玩家长时间位于该柱状体后方，则其临场感将大打折扣。

场景遮挡确定方案

当环境遮挡定义完毕后，如何对其实施有效的测试？如前所述，针对于视线的生成以及相关遮挡信息，光线投射技术可视为一类有效的解决方案，但需要注意的是，光线投射机制通常会产生较大的计算开销。若多数角色的相对运动速度已知，则无须在每次游戏更新循环操作中执行环境遮挡测试。同时，在视线的计算过程中，其开销可通过多帧分摊机制加以解决，当采用延迟技术时尤是如此。针对于对象的遮挡时机，一类有效的方案是根据连续遮挡帧的数量设置相应的加权阈值。其中，权值进而依据光线投射的时间或位置属性加以定义。同时，该类权值方案同样适用于相机在场景中的漫游操作。若光线投射无法抵达目标位置，则可将其视为一类遮挡信息并提示相机对此加以回避。光线投射的相对位置也可用于确定相应的躲避运动方式。

光线投射技术

当对应角色具备人类外观或类似形状时，可对 3 个位置执行可见性测试，即头部位置、躯干位置以及脚部位置。如果全部位置与相机之间均不存在可视视线，则可认为该角色处于遮挡状态。当然，还可对此执行进一步测试（例如 Z 缓冲区以及帧缓冲区分析），且硬件渲染信息通常会对绘制像素的数量提供相应的报告信息。这一类信息可清晰地展现当前角色模型的可见位置，进而可作为基于遮挡效果的衡量标准。

考虑到环境几何体以及玩家角色的复杂度，当执行相关遮挡测试时，单一的投射光线往往力不从心。通常情况下，角色的全遮挡效果往往并不重要，相比之下，角色部分内容（或角色的特定位置，如脚部位置）被遮挡往往会增加角色的定位或定向难度。针对此情形，角色的特定部位往往更具实际意义。综上所述，在某些方面，角色的可见性其重要程度不言而喻。另外，应尽量避免全遮挡现象的出现。针对于动画角色，建议视线位置应尽量靠近角色的平衡位置，亦即，角色的连接点位置通常不包含动画效果。

在游戏 Metroid Prime（[Metroid02]）中，第三人称视见画面中的玩家角色即采用了球体加以标识，而非人物模型。在此例中，视线计算过程将涉及球体上的 3 个检测位置，即球心、球体上端以及球体下端（基于世界坐标空间内的 Up 轴）。当全部位置的视线测试均失败时（或一段时期内），则当前角色可视为处于遮挡状态。该例子表明，应先期执行全遮挡测试并于随后通过相机系统执行其他相对显著的行为动作；若玩家可清晰地理解角色、场景环境以及控制参考帧之间的相互关系，则部分遮挡效果

通常并无任何不妥。

部分遮挡效果可作为相机系统的提示信息加以使用，进而使得相机可作出较为迅速的反馈动作（或采取某种较为适宜的方式）。实际上，部分遮挡的相关特征直接确定了其对应的处理方式，例如，若角色的头部位置被遮挡，则应适当地降低当前相机位置。

空间体投射

有时，相对于潜在相机位置，几何体复杂度可导致光线投射方案返回错误的计算结果。一类替代方案是将空间体投影至游戏场景世界中。不难发现，该方案将产生较大的处理器开销，因此，可能的话，应尽量避免采取这一类措施。另外，相关检测条件亦会导致相机出现镜头跳转，进而使得相机移动呈现为不连续状态。相对于某一新位置，空间体投射技术常用于对相机实施重定位操作。例如，玩家角色在经过一段连续的移动后，通常会执行镜头跳接操作，且根据玩家角色的移动方向，将第三人称相机置于其后方某一位置处。因此，此处应确保新相机位置位于游戏场景范围内，且玩家角色未被遮挡。鉴于光线投射技术并非万无一失，因此空间体投射方案则更为适宜。当然，若玩家角色的新位置已被预置，且随后的初始位置也采用类似的方式加以确定，则此方式可有效地避免一系列复杂的测试行为。如前所述，光线投射技术将会产生较大的处理器计算开销，而空间体投射方案则有过之而无不及。鉴于此类原因，在空间体投射过程中，相机常被视为球体对象。

渲染技术

针对于遮挡测试，另一类方案则通过目标平台的渲染功能提供玩家角色或目标对象的可见信息。该技术可通过多种方式加以实现，且随着渲染硬件性能的不断提升，此类技术日趋流行。下列内容列举了较为常见的解决方案：

- ☑ 采用固定着色技术并将标识符视为颜色值。该技术类似于计算场景世界的可见数据集时所采用的方法，亦即，从任意给定位置计算游戏场景的潜在可见部分，其中，用于测试的相关对象可通过固定着色亦即标识颜色值加以渲染。这里，环境几何体常以不同的颜色值加以渲染，并便于区分其他对象的像素数据。随后，可对显示缓冲区进行解析，进而针对相关绘制对象的对应像素数据加以判断。最终，属于该对象的绘制像素数量或分布状态可用于确定相应的遮挡范围。
- ☑ Z 缓冲区（深度缓冲区）分析。针对渲染当前场景后所生成的显示缓冲区，如果渲染系统可对此加以访问，则可相对于某一特定游戏对象获取其 Z 缓冲区信息，如目标角色。通过比较 Z 缓冲区绘制之前后内容，还可进一步得到对象的轮廓信息。为了进一步降低性能开销，此处也可采用对象的渲染边界信息对 Z 缓冲区的局部区域进行分析。
- ☑ 基于渲染硬件的遮挡查询。某些渲染硬件或软件 API 提供了自动检测方式，并根据深度缓冲区比较机制来判断渲染图元或模型是否可实施进一步渲染操作，读者可参考 [Sekulic04]以获取更多内容。相关信息可判断当前对象是否处于完全遮挡状态，抑或部分像素未通过深度缓冲区比较测试。

遮挡预测方案

针对上述遮挡问题，对当前环境进行预测可视为一类较好的处理方案。在实际操作过程中，考虑到游戏状态多取决于玩家的行为动作，因而相关方案颇具技巧性。当然，先期运行全体模拟环境并对当前游戏的未来状态加以预测则显得不切实际，但对玩家位置或其他对象进行预测则切实可行。

针对于未来行为（或位置）的预测技术，其对应实例是预测相机；相应地，反馈相机则使用目标对象的当前（或前次）状态。由于预测相机可对相机的漫游行为进行较好的预测，因而该类型相机则更值得关注。然而，在实际操作过程中，考虑到目标对象的具体特征，完美的预测机制通常难以实现。因此，在多数情况下，当确定目标对象的未来位置时，对其前次运动状态加以推测依然足够。需要说明的是，该技术主要出现于 AI 范畴，尤其是基于目标引导的对象位置计算。

预测视点位置

如果目标对象的移动状态通过物理模拟方式加以实现，则应针对该对象的运动属性提供足够的数据信息，并据此推断当前对象以线性方式移动。然而，对象很少呈现为直线运动状态。特定的玩家对象常根据其他对象（如“锁定”功能或盘旋移动）、受限路径抑或其他方式实现自身的运动方式。这里，由于目标对象的连续移动常可通过最近移动状态加以推测，因而维护对象的前次移动属性通常十分有效。有些时候，对当前游戏环境的熟悉程度将有助于实现目标对象未来位置的准确预测，当采用固定路径时尤是如此（例如，规定玩家在预定义方向上移动）。

通常情况下，很少对预测位置照搬使用。相反，当前位置与预测位置之间的偏差位置常作为首选方案。关于偏差量的确定，一类方案是采用目标对象速度与阈值比较后的结果。

视线

通常，遮挡与视线（LOS）往往具有不同的含义，但二者之间存在着千丝万缕的联系。当然，根据目标对象的物理特征以及游戏机场景几何体的复杂度，实现的真实含义也不一而同。多数时候，可按如下方式对 LOS 加以考察，即源于相机的投射光线是否可到达目标对象处，且不与静态（或动态）场景环境几何体产生任何碰撞。遮挡测试常会涉及当前场景环境中的诸多游戏对象，由于玩家常在这一类对象中移动并实施交互操作，因而相机的移动难免会出现问题。另外，某些玩家角色类型还将依据当前角色状态采取多光线投射机制，如前所述，此类信息多用于检测角色的遮挡效果。同时，视线测试多见于 AI 领域，当确定潜在目标对象时，其测试开销通常较为低廉。鉴于 AI 逻辑的更新频率较低，因而光线投射可通过多个游戏更新步骤加以分摊。

除此之外，视线计算也可用于碰撞检测处理过程。例如，针对相机而言，视线计算常用于判断某一对象是否脱离当前场景环境；抑或，玩家角色与相机之间的门对象是否处于关闭状态。

当计算 LOS 时，有时需要根据遮挡因素适当地调整相机的躲避行为。为了有效地处理 LOS 问题，可考察相应的遮挡因素——动态游戏对象抑或静态场景世界几何体，进而相机可采取瞬移或其他移动

方式。

该方案稍显直观，但视线的定义可根据游戏类型或角色和环境特质产生较大的变化。

这里，是否存在源于相机位置处的单一光线投射至目标对象，且与场景几何体（静态或动态）或其他对象之间不存在任何相交？该问题可视为处理 LOS 问题时的一类基本方案。该过程将会产生某一近似结果，但多数情况下，该结果依然足够。然而，不久将会发现，当目标对象定义为动画角色时，单一投射光线将使得对应测试过程变得捉襟见肘，其原因在于，随着动画操作的不断进行，当前角色范围将产生较大的变化。即使角色采用球体加以描述（如游戏 Metroid Prime），其 LOS 处理过程也需要得到多条投射光线的支持。关于环境遮挡问题，前述章节对此提供了多种可选方案，这一类方案也可用于 LOS 测试。

处理视线问题

相机可能会面临以下情形，即为了维护与玩家角色之间的 LOS，相机无法实现快速移动。当相机碰撞体受限于几何体中时，通常会出现类似情形。针对于此，许多相机系统均未采用碰撞体技术。自然地，这一类相机将会尝试各类机会以避免脱离当前场景环境或与几何体产生相交。LOS 的缺失可视为一类较为严重的问题，是否存在对应的可选方案以有效地处理这一类问题？下列内容将针对各种可能性进行逐一考察，且各类方案均包含了不同的美学特征。

预测方案

防患于未然，天下诸事大多如此。通过预定义路径或位置对相机位置实施完全操控时，可确保不会产生遮挡现象；抑或，在可接受的限制范围内，采用部分遮挡方案。而针对动态相机方案，上述内容还存在其他可能性。可以肯定且较为重要的一点的是，相机将根据其目标对象实施预测方案，这将极大地影响相机在游戏场景环境中的漫游操作。

图形变化

在某些场合下，下列情形亦可令人接受，即允许遮挡效果的存在，但相机不会对该现象产生任何反馈动作。该方案的优点之一便是，相机以相同方式实施移动和反馈操作，且针对游戏的其他元素亦是如此。此处，其连续性值得肯定，但当角色处于遮挡状态时，玩家如何对其实施操控？针对于此，应确保该角色的可见性，即使该角色位于几何体或其他游戏元素后方。相应地，存在多种图形处理方案可用于辅助玩家操作。首先，可移除遮挡几何体或调整其渲染策略（如采用透明度）以保持当前玩家角色的可见性。其次，玩家角色可在遮挡几何体的前方加以渲染，其实现方案稍有不同。例如，可仅渲染对象的轮廓特征。游戏 Super Mario Sunshine[Mario02]便采用了此类方案。同时，该游戏还采用了其他辅助性的图形元素以强调当前角色位于几何体的后方（例如，淡化角色周围圆形区域外部的屏幕元素）。另外，还可通过其他图形元素（如 HUD）、粒子系统等内容标识玩家角色的当前位置。除去位置数据之外，角色的方向辨识过程同样十分重要。由于玩家自身可见，因而第三人称视点稍显奇特。针对于此，可低调地添加某些辅助性视觉元素。

获取期望位置

相机期望位置常依据游戏类型以及相机的标准属性加以界定。当试图获取相关位置时，可采用某种机制对各种可能性加以评估，并根据美学以及游戏体验需求加以甄别。当然，位于游戏渲染几何体之外的相关位置，抑或被角色所遮挡的位置则不在考察之列。

一种常见的技术是，依据某种方式移动预测位置，使其貌似围绕当前垂直轴进行旋转。其中，该垂直轴由玩家角色位置处的世界 Up 轴所构成。另外，相机位置半径可保持固定抑或处于变化状态，例如，可采用相对于玩家角色的相机高度值。另一种方案则使用了基于可见性的八叉树信息，其中，潜在可见集将生成类似的位置数据。一旦位置数据确定完毕，相机将即刻实施重定位操作，亦即瞬移操作。

瞬移/跳转操作

相机的即刻重定位操作（瞬移）是一类解决 LOS 的常见方案。相机的新位置可通过多种方式加以确定，需要注意的是，该新位置应保持一定的合理性并便于玩家理解。在电影技术中，瞬间移动以及重定位操作通常称作镜头跳转。针对相机跳转操作，一类较为重要的既定规则是“30° 规则”，[Hawkins05]以及[Arijon91]对此进行了较为详细的描述。该规则表明，镜头切换完毕后，应确保相机位置和方向变化大于 30° 角。因此，该变化时的观察者能够感受到更为流畅的视觉画面，以及非简单的相机“跳动”效果。当跳转操作导致当前相机沿前次相机视见方向移动时，通常不建议使用该规则，无论移动方向朝向或背向目标对象。当然，该情形应确保玩家能够正确地理解当前画面所产生的变化。这里，可适当地添加某些视觉元素以强调场景所产生的变化，这也是一类较为常见的操作方式，如闪烁效果或高亮效果。如果这一类效果与场景变化同步发生，自然地，玩家将会感觉到某些重大事件业已发生。

另外，一类较为常见的方案是沿当前 LOS 向量移动相机（为了保持场景中相同的 Z 坐标位置，该操作可能会以 2D 方式进行），直至到达最终的有效 LOS 位置。该测试易于实现，例如，可在场景中由当前相机位置至目标对象逆向投射光线。一旦得到该光线的碰撞位置，即可将其视为此光线上的新位置。该新增位置将出现于同一光线（光线）之上，并位于背离碰撞点的有效距离范围内。因此，相机的近剪裁面或碰撞空间体（若存在）将不会呈现为碰撞状态。

保持控制方向

当计算期望位置时，必要时，应针对玩家保持相同的控制参考帧，这一点十分重要。其原因在于，相机快速的重定位操作将导致玩家暂时失去方向感，因此有必要保持场景画面的稳定性。相应地，玩家不得不对其角色进行适时的调整以保持当前控制状态。当出现该情形时，轻则，玩家会产生某种混乱感；重则，当前角色可能会被移至场景中的危险区域。第 2 章曾对此进行了较为深入的探讨。

生成路径

另一种可能方案是，通过程序生成相机移动路径，直至遮挡现象消除为止。由于相机与玩家角色

之间的相对位置千变万化，且场景复杂度也不尽相同，因此该任务实现起来较为困难。另外，生成后的路径应避免与当前环境和其他对象发生碰撞（且不应与近剪裁面相交），除此之外，相关路径还应满足一定的美学需求条件。下面将对生成路径进行简要的分析。

跟随角色移动

从理论上讲，可参照玩家角色人物的近期路径确定相机的移动路径。虽然有时无法保证相机的位置位于当前角色的上方或后方，但针对玩家而言，生成后的路径应具备一定的“合法性”。注意，该技术需要以某种循环方式对一段时间内的玩家位置数据进行存储。这里，相对于最近一次所存储的顶点，仅当超出预定义移动距离阈值时，方可添加新顶点。一旦获取了玩家移动记录，即可据此令相机沿该路径行进。然而，顶点位置可能并不符合相关美学条件，尤其是接近对象表面时。针对于此，可从这一类“面包屑”型玩家位置处分别投射光线（或尝试使用其他方式的测试操作），以获取符合相关美学条件的路径数据（例如，针对碰撞几何体采用垂直纵向检测并以此判断渲染几何体的接近程度）。如果存储位置相对于当前表面呈凸起状态且仍位于玩家角色碰撞空间体内，则无须执行该操作（其结果依然有效）。如果通过这一类路径确定相机的移动方式，其位置间隔可能过于稀疏并导致相机呈现为不连续的运动状态。除此之外，还可针对存储位置使用较为简单的插值计算，进而获取相对平滑的移动路径（尽管该方式可能会受限于快速的方向变化）。针对于样条路径生成方案，当使用存储位置作为控制点时，上述效果还可得到进一步的改善。

凸起地形

对象的凸起部分可视为一类较为复杂的遮挡现象。这里，“凸起”是指场景几何体的延展部分，针对玩家而言，可在其上行进；当从游戏相机角度观察时，当玩家位于凸起部分下方时，则该玩家将被遮挡。类似的情形还会出现于悬崖这一类地形中。当玩家的移动速度远远大于相机的移动速度时，或者玩家位于相机无法抵达的位置时，抑或相机在途经角色时执行了 180° 旋转操作时，通常会发生上述遮挡现象。

大多数相机系统通常对此束手无策且往往受限于这一类凸起地形。通常，该现象可通过 LOS 检测机制加以解决，并瞬间将相机移至非遮挡位置，且在该过程中对控制参考帧加以调整。另外，一类更为实用的方案还会涉及遮挡预测位置，且相机呈现为连续微进状态并行进至更为合理的位置处，其间，可对控制参考帧加以微调。

针对于这一类问题，首先需要对事发处进行检测。针对于此，可采用脚本相机方案加以处理，当然，通用方案往往更具优势。下列内容列举了相关的可选方案：

- ☑ 基于凸起地形的预定义脚本方案。该方案的处理过程则较为耗时，且需要设计人员确定并控制游戏相机的行为动作。游戏体验需求多与这一类预置相机方案相悖。
- ☑ 针对难以处理的实例，则采用预定义方案，尤其是封闭空间。必要时，可采用静态相机或样条路径以确保相机可实现移动和自身定位。通过限制自定义方案的数量，该过程将变得更为切实可行。

- ☑ 针对相机与目标对象之间的“地面”，如果 LOS 测试失败，则将相机瞬移至新位置。如果环境表面通过材质类型加以标记，则“地面”的确定过程将变得十分简单。当然，该操作过程也可通过表面法线加以实现。作为一类测试条件，表面法线数据并非足够，对此，表面标记也存在自身的局限性。这里，可采用玩家角色距地面的高度值作为测试条件，该方案适用于各类应用场合。
- ☑ 使用移动延迟功能以向玩家或目标对象提供近似路径。该方案可用于生成不同形式的相机路径，直至生成基于目标对象的 LOS 为止。考虑到玩家角色很可能呈现为连续移动状态（且方向各异），且环境几何体亦不断变化，因而相机路径的生成过程并非易事。
- ☑ 查询环境几何体并根据前次位置动态选取路径，且在该位置处，玩家与相机之间保持可见。此处，该位置将提供与凸起地形近似边相关的相机系统信息，并可用于确定相机路径。在路径的生成过程中，相机不应距几何体过近，以避免与近剪裁面相交，进而导致出现不和谐的渲染结果。除此之外，在相机的移动过程中，应避免相机出现垂直定向这一现象；同时，还应适当地降低转向幅度（亦即，围绕世界 Up 轴的快速重定向操作）。

回避纵向柱状体

如前所述，凸起地形可视为一类障碍物，且相机应对此加以躲避。这里，纵向柱状体也将会出现类似的问题。如果封闭空间内设置了大量的纵向柱状体，则问题将趋于复杂化。其中，玩家角色常会处于遮挡状态，且柱状对象间的漫游操作将变得十分棘手。此处，柱状体对象的不同之处在于，玩家角色途经柱状体对象后方时（短期内），该角色将处于正常的遮挡状态。此时，如果玩家围绕柱状体对象移动其无须相机逆向反馈即可见时，则场景画面令人愉悦。相机相对于遮挡现象的反馈操作和当前场景之间存在着较为微妙的平衡关系，其中，玩家角色仅处于暂时遮挡状态。一般地，由于角色的未来位置可用于确定遮挡状态，因而预测相机可有效地处理这一类问题。

预计算方案

在诸多方面，预测相机方案极具吸引力，其中之一便是相机的运动和方向确定方案。相机移动的预定义路径可视为一类手工方案，可对其进行测试并确保玩家在任何位置均保持可见性。当然，这一类路径的生成过程亦十分耗时。此处，若不采用全路径预定义方案，则相机的有效位置可通过离线方式加以确定，并在运行期内对位置数据实施插值移动计算。类似于线性路径方式，可对相机采取适当的限制并使其保持应有位置，该位置经计算后将产生较为清晰的玩家角色或游戏体验元素的视觉画面。在第 9 章中将会看到，这一类位置数据可视为空间体（通常为圆柱体或球体），因而相机位置的设置将更具灵活性。

防止丢失 LOS

如果无法保证 LOS 的存在（根据环境复杂度或游戏体验需求条件），则应全力避免这一现象的出现。

瞬间移动

相应地，可瞬间将相机移至某一新位置并使 LOS 可见。当选取新位置时，应注意保持同一控制参考帧以尽量保持玩家的方向感。尽管瞬时重定向操作常被人所诟病，但对于全遮挡现象，该方案依然存在一定的应用价值。

淡化遮挡几何体或对象

该技术可视为一类常见的选择方案，当然，具体操作过程还取决于当前环境或游戏对象的 Alpha 混合能力。通常情况下，淡化对象多保持部分可见，这将导致画面丧失其原有的和谐性。这一类美学问题表明，任何细小的疏忽都将对游戏场景画面产生负面影响。根据日常生活经验，实体对象通常很少呈现为透明效果。虽然该方案可满足当前游戏体验需求，但这也提醒玩家他们并非真正地身处于当前游戏场景中。同时，位于淡化对象下方或后方的场景几何体应实现为基于实体表面的封闭对象。

无须渲染遮挡几何体或对象

这里，针对于遮挡问题可采取一类较为简单的处理方案，即无须对遮挡对象实施渲染操作。然而，场景部分内容瞬间消失将使得画面失去应有的自然感；另外，若玩家角色在遮挡对象后方往复移动时，应确保该角色在渲染对象和非渲染对象之间保持适宜的变化速度。针对于淡化几何体，上述操作同样有效。当玩家遮挡状态发生变化时，可设置一个计时器以防止产生上述现象，其中，在既定时间范围内，对象的状态保持不变。

渲染图形元素且忽略遮挡现象

该方案涉及对象轮廓或其他表现形式的渲染操作，进而显示叠加于遮挡几何体上的目标对象。如果玩家角色被遮挡，此处可覆盖一幅基于该角色的二维图像，进而指明其在当前场景中的相对位置。然而，考虑到该方案将从游戏机场景中移除玩家，因而其临场感大大降低且难以满足相应的美学需求。在少数情况下，上述渲染形式可用于某些特定的科幻型游戏。例如，间谍或冒险类游戏中的 X 光技术，抑或科幻类游戏中的穿墙透视等魔幻效果。

可见性预计算方案

在多数游戏中，常可通过相关数据结构对游戏场景的渲染操作进行优化，或者通过 AI 可控对象实施场景的漫游。鉴于这一类数据结构常包含与场景划分或对象间空间关系相关的数据信息，因而可用于向当前相机系统提供漫游信息，如二分空间划分树（BSP）以及可视树。这一类数据结构多用于碰撞检测以及渲染操作且可实现系统间的共享。[Ericson05]以及[Eberly04]对此进行了较为详细的讨论。

从较高层次来看，当产生安全测试条件后，可视信息即可用于确定相机的新位置。AI 路径搜索数

据可用于定义场景内的相机路径，当然，该操作常会根据相机与几何体表面之间的接近程度定义相应的辅助约束条件。

安全检测

顾名思义，安全检测可视为一类问题处理方案，并根据相关美学条件或游戏体验需求对问题采取迅速处理。游戏相机的最大问题是遮挡问题，其次是如何避免几何体的遮挡现象。其中可能会出现下列情形，即相机止步于某一位置且无法提供适宜的游戏场景画面——应对此类问题做出迅速调整。另一类情形是，玩家角色跨越沟壑后相机却不能及时跟进；或者，门对象在相机与其目标对象之间呈现为关闭状态。

需要说明的是，在每次更新操作步骤中，无须对全部安全条件进行检测，且可将其划分为 3 类，即逐次更新操作、频繁更新操作以及少量更新操作。

逐次更新操作

某些问题不应超出当前游戏更新范围，因此，需要在各单次更新操作中对此类问题加以检测。

- ☑ 几何体穿越相机近剪裁面。该现象多发生于静态环境场景或动态对象之上，进而对渲染效果产生负面影响。例如场景中出现多边形边，或者当前游戏场景封闭空间的外部区域暴露于视觉画面中。此时，需要重新计算相机位置并将相机瞬移至该新位置处。另外，相对于距玩家角色的距离值、高度值以及控制参考帧的一致性，若满足相关条件，之前的有效位置数据依然可用。显然，应尽量避免出现该现象，并可通过增大相机碰撞空间体得以实现（或在相机移动过程中采取适宜的碰撞检测机制）。
- ☑ 无效的相机转换矩阵，该问题源于累积旋转操作或向量计算（如非标准化向量）所导致的数值误差。相应地，当相机围绕其前向轴执行旋转操作时，则无效操作在所难免。这一类问题可通过限制相机转换中的旋转内容部分加以解决。这里，可尝试对矩阵问题实施标准化操作抑或修正操作（如重构当前矩阵）。另外，也可将旋转矩阵简单地还原至最近有效矩阵。考虑到需要对该操作实施逐帧检测，因而场景方向可能会少有不同，其原因在于，两个渲染帧之间的间隔过于靠近。在场景的构建过程中，若转换操作存在不合理之处，应尝试对相关矩阵加以修正。例如，可将现有矩阵还原至单位矩阵（包含平移部分）。若参考帧有效（如第一人称游戏中的玩家方向），则可通过该参考帧构建正确的矩阵。
- ☑ 场景外部相机。如未做特殊说明，相机不应移至场景几何体外部区域。这里，可通过简单的测试判定相机是否位于当前游戏场景外部区域。其中，可将相机瞬移至前次安全位置处，抑或通过计算方式产生某一适宜位置。某些时候，这一类属性将变得异常重要，因而相关方法应内建于相机脚本系统中，并在恰当的时候对其加以覆写。
- ☑ 单一更新操作中围绕世界 Up 轴的过量旋转。当相机的 Up 轴几乎平行于世界 Up 轴时，则有必要在每帧中对旋转量加以限制。考虑到相机有时会围绕某一轴进行快速旋转，该效果有时亦称作转向操作。该操作通常产生于下列情形，即相机目标对象途经相机的上方或下方位置处。

频繁更新操作

鉴于某些问题并不会产生渲染以及其他负面效果，因而可在多帧中持续存在。然而，一段时间后，这一类问题也需要根据游戏体验需求条件进行相关处理。

- ☑ 指向目标对象的 LOS。通常情况下，LOS 应限定在目标角色的躯干或腿部位置。
- ☑ 距期望位置的距离。相对于期望位置，如果相机移动时的滞留时间过长（可能由位于相机及其目标位置之间的碰撞几何体所导致），则该状况表明，相机无法到达其目标位置。另外，目标对象的尺寸可变得相对较小（从当前相机位置处观察），以致无法对其实施有效的操控。然而，在多数情况下，LOS 测试可捕捉到上述非正常事件，因而执行这一类辅助性测试不失为明智之举。

少量更新操作

最后，在较长时间范围内，某些问题仅会影响到相关美学效果或游戏体验元素。当然，时间范围的定义并非一成不变且与游戏类型相关。在多数场合下，该范围不过几秒而已，但这已然足够。亦即，相机可在此时间段内处理上述问题，且不会出现瞬移现象。

- ☑ 玩家角色的有效遮挡（仅对第三人称相机有效）。如前所述，针对玩家而言，清晰的视见画面将有助于对其角色进行观察和操控。
- ☑ 相机距目标对象过远。若相机位置距离其期望位置过远，则将导致目标对象明显小于其正常的屏幕尺寸；同时，视域将呈现为狭长状态，这两种现象都将对玩家的操控产生极大的影响。

安全检测方案

一旦安全测试条件被触发，常见的处理方案是将相机迅速移至既定安全位置。如前所述，通常情况下，这可通过跳转镜头加以实现。另一种方法则是逆向遍历位置数据表且令相机按此方式移动，直至相关位置满足可视条件。针对于后者，该方案无法保证其最优性以及相机前次位置的有效性（例如，对象可能呈现为连续移动状态）。

本章小结

本章通过多种方式探讨了游戏相机与 AI 游戏对象之间的相似性。游戏相机以及游戏 AI 对象常需要在复杂场景环境中实施漫游操作，该过程应对碰撞现象加以正确处理，并避免出现错误的移动方式，从而降低玩家的临场感。游戏相机包含其他诸多需求条件，且需提供基于玩家角色或其他目标对象的无遮挡画面。需要注意的是，在相机方案的制定过程中，应在游戏机场景环境的临场感以及玩家角色无遮挡镜头画面之间寻求某种平衡性，且不应降低玩家的游戏体验感。

第 9 章　移动和碰撞

在第 7 章和第 8 章中，曾介绍了与相机移动相关的大部分内容，即相机期望位置及其漫游方案。除此之外，与相机相关的最后一部分内容即是相机从当前位置至期望位置之间的真实移动过程，且与位置的定义方式无关。本章将主要讨论某一时间段内相机位置的移动方案，且应确保此类运动方式的平滑性和连续性。在相机运动过程中，有必要检测相机与游戏场景环境或其他游戏对象之间的相交状态（通常应避免这一类现象的发生）。该过程亦称作碰撞检测，但更为重要的是，如何避免碰撞现象。本章后续内容将对此进行深入讨论。

相机移动序列

待相机期望位置确定完毕后（具体内容可参考第 7 章），则可尝试并采用多种方式移动相机。相机的物理移动不仅包含移动速度，通常还包含加速或减速运动。这里，建议相机的移动以及速度变化均呈现为连续运动状态（即不存在突发性的位置跳转）。其中，相机的速度与目标对象的移动速度紧密相关，然而，目标对象突发性的运动以及变化不应体现于第三人称相机中（针对于第一人称相机，其视见画面常锁定于相对于玩家角色的某一位置处）。该问题令人进退维谷。当目标对象背离相机加速（以及高速）移动时，相机是否会无法跟上该对象的运动节奏？而且，当相机停止移动时，如何防止相机距离角色过近也是一项重要的内容；另外一项需要考察的相关事项则是约束条件，亦即，根据一系列的设计规范限制相机的移动方式，例如三维空间体或距目标对象之间的距离值。这里，可将相机的移动方式视为一类具有一定层次结构的约束条件集（即各个相关属性的组合），可适当地对此加以协调，进而令相机的移动方式满足处于变化中的各类需求条件。然而，当要求相机与目标对象之间处于相对合理的距离范围时，相关约束条件可能会与此类需求发生明显的冲突。

因此，如何定义相机的运动方式？这里，可将该问题分解为多个独立步骤，尽管某些步骤对相机行为并不具备实际意义。

- ☑ 确定期望位置。
- ☑ 限制期望位置。
- ☑ 朝向期望位置生成预测移动路径。
- ☑ 检测该移动方式是否或产生碰撞（可选项）。
- ☑ 处理碰撞问题（可选项）。
- ☑ 生成并验证其他移动方案（可选项，首次移动无效时）。
- ☑ 将相机移至新位置。

当然，上述各项步骤取决于相机的期望行为以及是否存在潜在碰撞问题。考虑到相机的移动方式多与目标对象相关（通常为玩家角色），下面将考察二者之间的关系对相机移动所产生的影响。

人物移动

目标对象的移动行为（如玩家角色）通常会直接影响到相机的运动方式，且多源自相机期望位置的变化。除此之外，角色的（快速、即时）重定向操作也将致使相机产生显著的运动效果。下面将再次考察玩家角色的运动方式对第一人称相机和第三人称相机所产生的影响。

第一人称相机

根据定义，第一人称相机通常锁定于基于玩家角色的某一位置处。在角色的行进过程中，相机与该角色彼此协同工作，且会偏移至相关位置处，如角色的眼部位置。

需要注意的是，根据当前视域以及视口的宽高比，相机的真实位置可能异于角色模型的“眼部”位置。当从第三人称视点切换至第一人称视点时将产生问题。如果第一人称相机的最终位置脱离于角色的“头部”位置，则切换操作将会终止于某一错误位置处（即使该切换操作呈现为平滑状态）。另外，若切换操作满足图形正确性要求（即相机止于期望位置处），则会产生即时跳转现象，且一旦切换操作完成，即会对第一人称相机的位置加以修正。第 10 章将对相关解决方案进行深入的讨论。

同时，玩家的屏幕表现手法其定位方式通常异于对应的玩家模型。类似的情形还包括玩家对象的比例尺寸，据此，玩家可与游戏其他元素实现良好的匹配。当渲染机动设备的驾驶舱环境时尤是如此。这也意味着，针对第三人称角色视见画面（如电影视频序列或多玩家游戏），应适当地添加某些模型或动画元素以对上述情形实现良好的支持。

当角色在场景几何体中漫游时，其纵向移动常会导致相机的垂直抖动。鉴于相机的非连续状态以及对玩家所产生的干扰，应尽量避免出现此类现象。该问题可通过下列方式加以解决，即在一段时间内向相机的纵向移动方向添加阻尼效果。在一定条件下也可禁用阻尼效果，以在相机和玩家之间实现正确的移动。相关实例包括外力作用下的玩家移动（如重力）、电影视频序列以及（第一人称相机与第三人称相机之间）切换操作。

第三人称相机

角色移动方式将以多种形式对第三人称相机产生直接影响。这里，首先需要处理的问题是延迟现象。

技术要点

为了有效地避免相机纵向噪声，一类常见的阻尼方案是限制每次更新操作中的移动量。下列代码显示了对应的基础方案：

```
float const kMaxZMotionPerSecond(0.25f);
//desired maximum motion
float zMotion = newPosition.GetZ()-oldPosition.GetZ();
float const maxZMotion = kMaxZMotionPerSecond * deltaTime;
zMotion = Math::Limit(zMotion, maxZMotion);
newPosition.SetZ(oldPosition.GetZ() + zMotion);
```

这一类平滑操作相对粗糙，其中，纵向移动量定义为常量（亦即，不存在加速或减速运动），这将在纵向移动的开始和结束处产生不连续移动。一类替代方案则使用了阻尼弹性方案或 PID 控制器。第 10 章曾对弹性机制以及 PID 控制器进行了较为详细的讨论，其常见的应用方式如下所示：

```
float const zMotion = newPosition.GetZ() -
  oldPosition.GetZ(); verticalSpring.SetLength(AbsF(zMotion));
// update the spring length
// Here the target spring length is zero, and need not
// be set each time.Typically the spring length is
// unsigned, so that must be dealt with.
// perform the convergence
float const newZMotion = verticalSpring.Update(deltaTime);
if (zMotion > 0.0f)
  newPosition.SetZ(newPosition.GetZ()- newZMotion);
else
  newPosition.SetZ(newPosition.GetZ() + newZMotion);
```

需要注意的是，当采用弹性或其他收敛方案时，新纵向位置则可通过下列方式加以实现，即计算当前弹簧长度，随后，可从目标坐标至当前值逆向使用该值。弹簧自身长度定义为无符号值，且仅表示距期望位置之间的距离。在大多数相机处理方案中，该技术同样适用，当采用严格意义上的阻尼收敛方案时尤是如此。

由于位置延迟技术将使得相机产生更为自然的观感，并着重强调角色的加速或减速状态，因而其应用范围较为广泛。同时，该方案与传统影视技术中摄像机的移动方式具有一定的相似性。亦即，摄像机的跟拍或移动应呈现为平滑的加速或减速状态。然而，若相机与玩家角色之间的距离过大，二者之间将失去原有的位置关系；同时，相机的场景漫游机制将变得难以操作并会增加对象间的遮挡几率。针对于此，应将延迟量降低至最小程度。

如前所述，相机的位置应保持在角色“后方”位置处，针对于此，玩家角色的快速旋转操作可能会导致问题的出现（如瞄准功能）。此处，相机多围绕圆弧运动，进而维持与当前角色之间的期望距离。

针对于第三人称相机，封闭场景环境将使得问题区域复杂化，尤其是玩家角色将相机置于角落处时。若几何体过于复杂，则相机甚至无法到达其期望位置处。几何体的避让问题涵盖了丰富的内容，稍后将对此加以单独介绍。

另外，玩家角色快速的加速或方向变化同样会对相机系统带来各类问题。鉴于相机的平滑移动特征，因而玩家的快速变化将会大大削弱此类特征。另外，若相机反应速度过慢，则有可能滞后于当前角色，或二者之间彼此相交。同时，若相机移动过快，则场景整体变化将呈现出一种混乱感。

如何保持相机移动的平滑性？如何消除上述问题所产生的负面影响？针对于此，限制玩家的加速状态可视为一类潜在解决方案，尽管该方案可能因某些游戏设计元素产生冲突。在某些情况下，可将相机移动与玩家进行绑定（当相机朝向玩家角色移动时，至少可避免二者之间的相交问题），但该行为可能会产生抖动现象。

需要注意的是，下列问题应及时加以处理，即相对于当前玩家角色，相机加速或减速过于缓慢。针对于前者，将导致移动延迟现象；而对于后者，则会出现画面过度等现象。

移动方案

下面将对某些较为重要的移动方案加以介绍，进而可在一段时期内对相机的位置进行调整。

瞬间移动

如前所述，此类移动方式将相机迅速移至场景内的某一新位置。待非交互式电影视频系列播放完毕后，且当对当前玩家角色实施重定位时，通常需要执行此类操作。实际上，相机的瞬间移动与镜头的跳转镜头具有相同的含义。当以该方式移动相机时，这也意味着，相关操作应遵循 30° 规则。然而，这一类美学条件通常在期望位置的定义阶段加以使用，而非真实的移动过程。

技术要点

当对相机实施瞬间移动操作时，下列内容应予以考虑：

- ☑ 根据相关确定方案，可从当前新位置处重计算视见位置。同样地，方向问题也可采用类似的方式进行重计算。
- ☑ 这里，应对目标位置的有效性加以检测，以避免相机距离场景几何体过近、位于对象内部等现象。当然，该检测过程很可能也是期望位置确定方案中的一部分内容。
- ☑ 可能的话，应保持前次控制参考帧，并在一段时间内对新参考帧进行插值计算。第 2 章曾对控制参考帧进行了较为深入的探讨。
- ☑ 此处，建议定义相关函数执行此项移动功能，并对某些常规性操作加以整合，且无须针对各应用实例直接更新相机的转换矩阵。某些相机需要使用到外部对象，且这一类对象将根据主相机对象进行移动。因此，若相机在当前游戏场景内执行重定位操作，则该类对象也据此执行相应的更新操作。
- ☑ 如果需根据当前相机速度计算多普勒效应，则应对最大速度加以限制，以避免相机快速移动时所产生的某些问题。

锁定相机

相机位置可锁定于期望位置处（即每帧均定义为期望位置值，即使相机发生显著变化或以不确定方式加以定义时），如前所述，这也可以视为一类瞬移方式。在多数时候，该操作具有明显优势（如路径相机），但在某些情况下则会出现问题。考虑到整体场景亦会产生相应的变化，因而至期望位置处的不连续移动势必会反映至相机的移动过程中。有时，可在单一轴向上将相机位置锁定于期望位置处，且该轴通常为垂直轴。

如果期望位置绑定于玩家角色之上，即使角色的轻微移动（或者方向变化）亦会在相机中有所体现。该操作多出现于第一人称相机中。在某些特定环境下，依然需要在纵向移动方向上添加阻尼效果，如玩家在崎岖地形表面上行进时。针对玩家角色的外部视野，当相机锁定于完全位置处时，其相对移动可能会出现抖动现象。另外，相对于当前角色，相机貌似固定于角色一端，因而显得极其不自然。最终，这将会加重玩家的混乱感，严重时，还会导致玩家出现晕眩等现象，当更新操作频率不一致时尤是如此。

在某些情况下，直接将相机移至目标位置处也是可行的。在封闭环境空间内，玩家角色不应超越

当前相机位置（或者与相机相交）。如果相机移动锁定于目标对象的某一位置处，则可有效地解决上述问题，但相机的移动易受到玩家角色移动的影响。除此之外，目标对象的速度可部分添加至相机中，以避免二者之间产生相交问题。另外，基于约束条件的路径相机则可有效地消除玩家角色所产生的抖动现象。将相机锁定至期望位置时应避免相机的滞后现象，并确保该相机不会与目标对象相交。这里，假设期望位置的确定方案中不包含此类问题。

在大多数第三人称相机中，建议在相机与玩家的反馈之间引入延迟机制。针对于此，方案之一便是使用比例控制器。

比例控制器

这里，可直接调整相机对象转换矩阵中的平移分量，以对相机位置实施更新操作。其中，可适当地添加阻尼或加速效果，但这一类效果常通过特定的方式加以使用（例如，物理模拟过程中不使用质量或摩擦力）。如果未向速度中添加阻尼效果，则一旦到达期望位置，相机将会戛然而止；抑或相机将会突然加速，从而导致运动缺乏自然感并产生不连续现象。

大多数游戏均采用比例控制器实现相机的运动。根据位置期望特征以及相机与期望位置或方向之间的角度关系，比例控制器可对运动量进行调整并反馈于当前相机系统。通常情况下，该过程可依据相机距期望位置之间的距离加以计算，因此，当相机接近期望位置时，其速度可以线性方式逼近 0 值，进而确保平滑的减速（或移动）行为。

技术要点

典型的比例控制器可按下列方式加以实现：

```
Vec3 const deltaPosition = desiredPosition -currentPosition;
// the distance at which velocity damping is applied
float const kDampDistance(5.0f);
float const K = Math::Limit(deltaPosition.Magnitude()/kDampDistance, 1.0f);
// Limit constant to 0..1 based on distance
Vec3 const cameraVelocity = deltaPosition.AsNormalized()* K;
Vec3 const newPosition = currentPosition +cameraVelocity * deltaTime;
```

在相机靠近静态目标位置过程中，上述移动方案包含了一些显著的特征，但该方案并不处理目标位置的移动行为。鉴于速度与距离之间多呈现为某种比例状态，当目标对象远离相机时，相机的移动在初始状态下呈现为平滑状态，但随后体现得更为明显。在最坏情况下，相机将被迫突然调整其速度，该问题可通过限制相机的加速度以得到某种程度的缓解。然而，考虑到无法实现及时减速，因而相机的最终位置通常会超出原期望位置。

技术要点

如前所述，相机滞后问题可通过以下方式加以解决，即向原公式中添加目标对象的速度数据。针对于此，可替换前述代码的最后两行内容，如下所示：

```
// unless we can obtain it directly from the object itself
Vec3 targetVelocity = (desiredPosition -previousDesiredPosition) /deltaTime;
Vec3 cameraVelocity=(deltaPosition.AsNormalized()*K*deltaTime)+ (targetVelocity * T);
```

需要注意的是，可按比例调整目标对象的加速状态并适当地调整目标速度（通过常量T），进而可较好地解决该目标对象的加速变化问题。类似地，若目标对象朝向相机自身移动，则可适当地增加T值。目前，可根据相机目标对象按比例控制相机的速度，但这将受限于目标对象的速度变化。为了实现平滑移动，需要在一段时间内对相机实现加速运动，并提供相应的限制方案以形成某种程度上的运动延迟。针对相机运动而言，延迟机制提供了更为自然的运动方式。

```
float const acceleration = Math::Limit( (desiredVelocity -
currentVelocity), kMaxAcceleration);
// may need an accel limit
Vec3 const currentVelocity += acceleration * deltaTime;
Vec3 const desiredPosition = currentPosition + (currentVelocity * deltaTime);
```

物理模拟

基于相机运动的物理模拟方案，其全面阐述则超出了本书的讨论范围，读者可参考[Eberly03]以获取更多内容。下面将对某些基本概念进行简要的介绍。

相机对象将根据牛顿定律或基于游戏对象受力方案的其他物理模拟形式处理其运动问题。通常，相机移动方式多通过刚体的速度和加速度方案加以计算，但一般不会涉及旋转力，且可采用直接方向控制以确保游戏对象的取景操作（或许，模拟人类行为可视为一类例外情况）。碰撞检测及其反馈动作使得相机更像是一种“浮动”于空间内的对象。然而，虽然阻尼作用常可降低由不规则几何体引起的抖动现象，但相机通常不包含重力计算。除此之外，相机还包含其他限制条件（如地形上方的高度值、与水面的结晶程度等）和/或期望位置计算过程中的某些限定内容（例如，距目标对象之间的距离值）。

一般地，物理模拟通常会产生较大的计算开销，但随着现代处理器的不断发展，其运行时钟频率不断刷新其原有记录，因而这一类模拟计算变得越发普遍。需要注意的是，当相机被视为一类物理对象时，应避免其他物理对象与相机之间产生彼此作用，即碰撞不应移动相机或改变相机的方向。针对于此，可禁止相机与其他基于物理驱动的游戏对象产生任何交互作用。通常情况下，其他对象不应过于靠近相机对象。该情形可通过多种方式加以实现，相机可向非玩家游戏对象（如残片或离子系统）施加相应的作用力。另外，相机还可对某些 AI 路径寻路算法产生影响。群聚算法以及躲避算法（如Boids）最初由[Reynolds87]提出，且使得游戏对象可对其他对象产生直接影响。因此，AI 驱动对象可通过斥力躲避相机，进而避免与当前相机相交。该技术亦可用于降低目标对象的遮挡几率。考虑到上述方案将对游戏体验产生较大影响，因而应谨慎使用。然而，深入了解（与相机相关的）游戏对象的各项内容将有助于设计出相对“真实”的游戏体验方案。如果相机位置处于玩家操控之下，则在游戏体验过程中，玩家可有意识地对其进行适当的调整。理想情况下，相机应仅以“观察者”身份出现于游戏中，相对于其他游戏对象，相机应具备相应的躲避功能。

如若希望相机能够沿碰撞网格表面滑行，则物理模拟不可或缺。如果这是一类期望行为，则实际操作过程还将取决于碰撞几何体的构建过程。此处，应确保相机可沿表面平滑移动。需要注意的是，相机有可能“粘贴”于碰撞几何体之上。当然，该情形并非是期望中的结果。通过构建合理的碰撞网格，上述问题可降低至最小程度。尽管出于性能考量，碰撞网格都会经过适当的简化，但应尽量对其进行调整并保持其“平滑”外观，以使玩家角色方便地在场景环境中移动。当不考虑物理模拟的使用方式时，该操作尤为有效并可方便地实现相机自身的平滑移动。本章后续章节将针对相机碰撞的物理模拟加以详细的讨论。

弹性机制（弹簧）

相机的移动往往会超出原期望位置，这也是一类较为常见的问题。当采用比例控制器移动方案时尤是如此，此时，目标位置向相机移动或突然停止（亦即，相机的减速过程慢于相机与其目标对象之间的距离缩短过程）。因此，相机将在其期望位置处产生遮挡。

针对于相机位置的遮挡现象，一类方案是通过弹性机制控制相机的移动，这将在一定程度上缓解或消除此类问题。当采用弹性机制时，如若不考虑其实现方式，弹簧的硬度应适中并保持游戏相机的自然状态。相对于目标对象，第三人称相机的杆状移动通常不可取，尽管其应用较为普遍。另外，相机、目标对象或位置的加速度或减速度应尽量保持平衡。除此之外，相机不应超出其目标位置；同时，当相机过于接近或远离目标对象时，还应避免相机出现突然停止这一类现象。针对这一类期望位置，相机移动可采用严格意义上的阻尼方案。

亦即，若不考虑相机的移动方式，相机应准确定位于目标位置处。在实际操作过程中，该方案通常难以实现，相信相关行业的从业人员对此深有体会（如温度调节器以及巡航控制系统）。弹性机制有时可通过 PID 控制器（稍后加以介绍）加以实现，并结合比例控制器以实现玩家角色的快速加速方案。由于该方案可实现较好的移动特征（基于目标位置的准确定位），因而可将部分玩家速度作用于相机速度之上，这一点与前述章节所提及的比例控制器类似。

第 10 章将介绍一类相对简单的弹性机制，下面则集中讨论弹性机制与 PID 控制器之间的匹配方案。

PID 控制器

PID 控制器常见于工艺控制行业，如流体控制值、温度调节装置以及制动系统等，并与控制器之间形成某种反馈机制，进而减少当前值与期望值之间的误差项。针对于此，可构建一类虚拟弹性机制，并与阻尼效果结合使用。

通常情况下，弹簧相对于其原始位置产生振荡，但这一类振荡效果对于大多数游戏相机而言并不可取。理想情况下，相关数据应尽量接近期望值（通常为 i 距离值，但也不仅局限于此）。如果控制器遵循上述行为，即可将其视为严格意义上的阻尼方案。相应地，PID 控制器一般可提供上述功能。第 10 章将对此加以详细讨论。针对于具体的实现过程，PID 控制器与弹性机制具有异曲同工之处。一般地，PID 控制器的设置点应与期望距离或角度值实现较好的匹配，且于每次更新操作中逐步接近目标值。在执行更新操作之前，有必要在 PID 控制器中设置当前值，以应对外部因素所产生的变化（例如，目标对象的移动改变了与当前相机之间的距离）。

技术要点

当确定相机与其期望位置之间的距离时，下列内容采用了 PID 控制器逻辑。第 10 章将对控制器的实现过程进行详细的讨论，此处仅考察其相关应用方案，如下所示：

```
Vec3 const currentDelta = currentPosition —desiredPosition;
float const currentDistance = currentDelta.Magnitude();
float const newDistance = PIDController.Update( desiredDistance, currentDistance, deltaTime);
Vec3 const currentPosition = desiredPosition + (currentDelta.AsNormalized() * newDistance);
```

其中，PIDController 中的 Update()函数包含 3 个参数，且分别表示当前设置点（或期望值，相对于前次计算，该值产生变化）、当前值以及自前次更新操作以来的时间间隔。这里，可使用返回后的距离值计算背向期望位置处的新位置。

圆周运动

这里，可将相机期望位置定义为相对于目标对象或其他参考帧的角度值，类似的方法也适用于相机移动方案。圆周运动可向第三人称相机提供全自由视角，且相机围绕玩家角色在水平圆弧内移动（可与纵向移动结合使用，以使相机穿越圆柱体或球体表面）。在多数时候，相机无须严格按照圆弧的方式进行移动。如果期望位置接近于当前位置，则线性移动可产生近似的移动效果。另外，相对于目标对象的前视方向，若相机需要维持于某位置处（仅以圆弧方式移动），则角位移也可视为一类有效的处理方案。

通过该方式，相机位移可作为向量间的夹角加以计算。其中，相关向量可通过下列方式加以构建，即参考点分别指向当前相机及其期望位置。

根据相机角速度减少位移即可定义相机的新位置。此处，存在两种方法可对角速度加以计算，即均匀角速度（CAV）以及均匀线速度（CLV）。

均匀角速度

当采用 CAV 时，相机将采用相同的角速度分量运动且与其旋转点半径无关（无论是否存在阻尼效果）。因此，相机将采用同一时间量完成全部旋转操作，且每帧将会覆盖较大距离范围并与旋转点半径呈现某种比例状态。

均匀线速度

CLV 每秒均移动固定距离值，亦即，弧长呈现为固定值且与旋转点半径无关。其结果为，在相机接近旋转点的过程中，相机将会呈现为快速移动状态。通常情况下，有必要对相机的角位移加以限制，进而避免弧长过于接近或小于移动距离时所产生的快速旋转现象。

另外，如果未曾采用加速或减速效果，则相机围绕玩家的角位移将会呈现出不连续状态，即使玩家对此采取直接操控，该情况依然无法得到有效的改善。无论如何，在相机接近目标位置过程中，比例控制器将有助于相机实现平滑的运动方式。

技术要点

针对均匀角速度的相机移动，其实现方法如下所示：

```
Vec3 cameraBearing = Vec3(currentPosition -referencePoint); // assumed not coincident
Vec3 desiredBearing = Vec3(desiredPosition -referencePoint); // similarly
float const radius = cameraBearing.Magnitude();
// may be known
cameraBearing.Normalize(); // assumes this is valid
desiredBearing.Normalize(); // similarly
float const angularMotion = angularVelocity * deltaTime;
// rads/update
Quat const q = Quat::ShortestRotationArcClamped(cameraBearing, desiredBearing, angularMotion);
// See [Melax00] for this
Vec3 const newBearing = q * cameraBearing;
```

```
// rotate the original bearing
Vec3 const newPosition = referencePoint +(newBearing * radius);
```

在上述代码中，应注意以下问题：首先，代码根据相机的当前旋转平面分别定义了变量 cameraBearing 和 desiredBearing；其次，代码使用了四元数对相机的新位置加以定义。需要说明的是，函数 ShortestRotationArcClamped() 的中心思想源于[MelaxOO]中的 RotationArc()函数。其中，RotationArc()函数将生成一个四元数，其效果等同于前两个参数之间的旋转操作。这里，四元数的优势在于，保持原有 Up 向量且兼顾 LookAt()函数所包含的旋转特征。除此之外，该函数还将旋转量限制在第 3 个参数所定义的最大角度值上。

均匀线速度的实现过程如下所示：

```
Vec3 cameraBearing = Vec3(currentPosition -referencePoint); // assumed not coincident
Vec3 desiredBearing = Vec3(desiredPosition -referencePoint); // similarly
float const radius = cameraBearing.Magnitude();
// may be known
cameraBearing.Normalize();
desiredBearing.Normalize();
float const arcLength = angularVelocity * deltaTime;
// units/ update
// since arc length = r * cos(angle) we can convert it
// back into an angle
float const angularMotion = acosf(arcLength/radius);
Quat const q = Quat::ShortestRotationArcClamped(cameraBearing, desiredBearing, angularMotion);
Vec3 const newBearing = q * cameraBearing;
// rotate the original bearing
Vec3 const newPosition = referencePoint +(newBearing * radius);
```

在上述两个例子中，必要时，可单独确定相机的纵向位置。该类型相机多用于控制水平面上的移动，且与纵向移动无关。若计划将相机移动限制在球体表面之上，则上述函数依然有效，仅期望位置的确定方案稍有变化。

插值计算

当试图调整相机的行为动作时，通常希望相机属性呈现为平滑的变化状态，其中，相机速度可视为较具影响力的元素之一。考虑到一段时间内相机可能会产生快速的位置变化，因而在某些情况下，平滑的插值计算并非易事。此处将考察两个相机速度之间的插值计算以及更为重要的加速度插值计算。

移动插值计算通常与两个路点或场景位置间的直线路径相关。虽然移动多呈现为直线方式，但针对一段时间内的距离值，相关计算过程并未采用线性方式。通过使用插值函数将时间值映射至相机间的位置数据（正弦曲线或三次多项式，如 Bezier 曲线），通常可获取非线性移动方式。一旦采用了非线性移动方案，则可对相机实施加速或减速操作，进而确保相机开始以及结束阶段的平滑运动。为了有效地避免插值开始和结束时的不连续状态，除了速度数据之外，加速度亦不可或缺。

因此，基于插值计算的相机间的移动并不需要一定呈现为直线方式。取决于具体的操作环境，可在当前相机位置以及期望位置间生成样条路径，并于随后在该路径上使用均匀速度（非线性）。当相机处于静止或接近静止状态时，该插值类型通常十分有效。常见的例子包括第一人称相机与第三人称相

机之间的切换操作，以及自由态第三人称相机与受限路径相机之间的切换操作。

鉴于严格意义上的线性移动不复存在，因而当前位置以及目标位置之间的插值计算也体现了运动过程中的难点所在。第 10 章还将对此类内容进行较为深入的探讨。

平滑效果和阻尼技术

相机运动中一项较为重要的内容即是确保相机的移动呈现为连续、平滑的状态。针对于此，限制相机的加速行为将会导致以下问题，即当前角色的加速度可能会大于相机的加速度；同样的情况也会出现于减速操作。相机速度通常采用平滑技术以及阻尼技术以保证其运动呈现为平滑状态，且与目标对象的移动以及加速状态无关。

阻尼方案通常具有移动的缓冲功能。相关方法将输入值（通常位于 0～1 范围内）以非线性方式映射回同一范围，以使该范围内首尾两处的曲线导数为 0。根据具体形状，这一类功能项有时也称作 S 曲线或突起函数（冲击函数）。

阻尼技术

少数情况下，针对相机的延迟方案或减缓相机移动时，应谨慎使用相关方案，以防止相机产生不连续运动或快速的移动变化，进而分散观察者的注意力。有时，这一类功能内建于期望位置确定方案或相机的移动属性中（尤其是期望位置处于不连续变化中）。此时，可根据各类外部因素适当地降低相机与期望位置之间的运动量，如与期望位置之间的接近程度。

然而，阻尼技术无须同步地运用于各个运动轴之上。纵向移动可视为一类较为常见的相机不连续运动方式。该情形多见于第一人称相机中。当玩家角色在场景世界中漫游时，常会遇到小型障碍物以及各种不同形式的场景几何体。此处，无须采取较为明显的上下移动方式对此加以操作（以匹配于玩家角色的移动），可向相机的垂直分量添加阻尼效果，即使使用了第一人称相机。此类阻尼方案即可应用于世界坐标空间内，也可出现于相机渲染转换空间中。

在某些情况下，作为一类视觉元素，剧烈的垂直移动或上下运动可用于强调玩家在场景中的奔跑感。针对某些玩家而言，这一类剧烈的运动方式可能会引发某种不舒适感，如果设计人员执意在游戏中设置此类视觉元素，则应向玩家提供对应的禁用功能。需要注意的是，该类相机移动方案并不会导致相机过于接近渲染几何体这一现象。

在第三人称相机中，剧烈的纵向运动常会导致玩家失去方向感，因而应适当地限制相机的垂直速度。针对于此，第三人称相机常会向其纵向移动方向添加阻尼效果，进而过滤掉期望位置处的细微震动。在某些情况下，垂直方向上的阻尼运动并不适用，或者更为准确地讲，需要添加额外的纵向运动。例如，处于移动状态下的运载平台即可视为一类相关实例。一般地，当目标对象受外力影响而处于运动状态时，如运载平台或电梯，影响当前相机施加同一外力，以保证基于相机平滑移动的相关计算可在目标对象的局部空间内进行。

技术要点

作用于相机之上的外力应与目标对象的受力彼此匹配，对此需加以谨慎处理。例如，可针对某一移动路径或其他位置决策方案应用同一位移方案。一类较为常见的方法是令相机记录与其目标对象相关的位置信息，进而可将目标对象的任何移动数据复制至当前相机中或其相关期望位置处。当然，相机的更新操作应位于目标对象之后，以确保位移值可得到正确的计算结果，这一点十分重要。

移动过滤器

除去阻尼技术之外，还可对相机移动采用其他形式的过滤机制。例如，可通过过滤器移除相机移动过程中所产生的噪声或细微震动。某些时候，噪声则源于游戏控制器、物理模拟误差所导致的玩家轻微移动，抑或期望位置决策方案所产生的各种变化。无论如何，这一类细节变化均会在相机的移动过程中有所反映。一类常见的方案是使用低通过滤器，并根据既定频率阈值移除无效移动。更为复杂的方案还将存储近期相机的移动记录，并根据这一历史数据确定期望移动行为。需要注意的是，此类过滤器均会在不同程度上降低相机的灵敏度，如果相机运动超出阈值范围，或许还会产生细微的不连续现象，因而应对其加以谨慎处理。

某些类型的移动方案已经包含了相应的过滤或阻尼技术元素，但即使如此，考虑到外部因素所产生的影响，采用独立于相机移动方案且具有层次结构的过滤机制依然十分必要。

技术要点

下列内容实现了一类相对简单的过滤器：

```
Vec3 const movementDelta = newPosition -currentPosition;
if (movementDelta.Magnitude() > kMovementThreshold)
{
  currentPosition = newPosition; // a very crude fi lter
} else
{
  // ignoring small changes may cause the camera never to
  // move at all
}
```

更为复杂的过滤器则采用（经过滤操作后的）最新位置并将其视为下次决策方案中的部分内容，进而形成某种反馈机制。因此，也称作反馈型控制器。第 10 章中将要讨论的 PID 控制器便是一例，除此之外，其他类型的控制器还包括有限脉冲反应滤波器（FIR）以及无限脉冲反应滤波器（IIR），二者在数字信号处理过程中均十分常见。第 10 章将对此进行相似的讨论。下列内容显示了上述过滤器的使用方式：

```
Vec3 const movementDelta = newPosition -currentPosition;
if (!close_enough(movementDelta, Vec3::Zero())
// must be valid movement
{
  float distance = movementDelta.Magnitude();
  // desired distance
  distance = mMovementFilter.Update(distance);
  // IIR or FIR fi lter, etc.
```

```
    currentPosition + = movementDelta.AsNormalized() *distance;
}
```

类似的滤波方案也可用于相机速度之上，而非相机的移动距离。

移动约束条件

在某些场合中，有必要对相机的定位方案加以限制。通常情况下，其目的在于相机与渲染几何体相交后所产生的负面图形效果。有时，应保证特定游戏体验元素的可见性（或隐藏状态）；抑或确保相机位于当前游戏环境边界范围内。有时，这一类约束条件可应用于相机期望位置的决策方案中，且常称作软约束条件。然而，某些情况下，在不影响期望位置的前提下，相关约束条件可有效地阻止相机的移动。针对于此，待全部位置确定方案以及移动过程执行完毕后，方可实施上述约束条件，以确保最终的相机位置处于约束范围内。这一过程常称作硬约束条件。

纵向移动约束条件

这里，对应示例显示了作用于第一人称相机或第三人称相机的硬移动约束条件。如果玩家可潜入水面之下或其他渲染表面网格，则表面几何体不应与相机的近剪裁面相交。当出现此类情形时，画面将显示某些剪裁后的多边形边，这也使得整体画面显得十分不专业。另外，如果采用全屏渲染效果模拟相机的临场感，当相机的期望位置在水面交接处快速地往复移动时，可设置相应的开启/关闭切换功能以避免出现上述现象。当然，该问题也可通过下列方式加以解决，即防止相机过于接近水平面表面（在垂直轴向上）。相机应维持其垂直坐标数据，直至期望位置移出当前范围之外。由于相机将切换至某一新位置处（即从平面上方位置移至平面下方位置，反之亦然），因而这将导致一帧中的不连续运动状态，但针对相机与水面相交产生的负面渲染效果而言，上述方案依然值得推荐。

针对当前特例，可令渲染水平面表面远离当前相机以确保消除移动中所产生的不连续状态。

技术要点

以此方式阻止相机的移动将会产生一系列的约束条件，必要时，可根据相应的优先顺序加以使用。移动限制条件通常用于消除具有一定负面效果的图像画面，如几何体相交，该类型限制条件不同于漫游约束条件，针对于后者，其约束条件将对相机的当前位置产生直接影响，而非期望位置。针对水平面约束条件，其实现方式如下所示：

```
float const kMinimumZDistance(0.1f);
// as close to the plane as possible
float const zDistance(currentPosition.GetZ()-waterPlane.GetZ());
// i.e. from the plane toward our position
if (fabs(zDistance) < kMinimumZDistance)
{
  currentPosition.SetZ(waterPlane.GetZ() +
  Math::Sign(zDistance) * kMinimumZDistance);
}
```

再次强调，针对前述阻尼方案，可从当前平面至相机当前垂直位置并采用类似的方法执行逆向操作。其差别在于，

此处需要维护相关约束条件。一旦期望位置远离对象表面，将在每次更新操作中产生不连续运动状态。由于相机需穿越当前平面以到达相应的新位置处，因而该问题无法避免。当与某些图形效果结合使用并隐藏瞬移效果时，该问题可在一定程度上得以缓解（例如，针对此例而言，可在渲染场景上添加 2D“飞溅”特效）。这一类约束条件可视为平面约束条件的一个特例，具体内容将在稍后加以讲解。例如，游戏 MetroidPrime 系列即采用了此类移动约束。

渲染几何体的接近程度

另一类潜在的问题是，相机距渲染几何体过近。再次强调，视锥体近剪裁面与渲染几何体相交将会对整体画面带来负面效果。针对于第一人称相机场景，角色的碰撞体通常大于近剪裁面与第一人称相机所形成的空间体，因而可有效地避免这一问题。然而，根据视域以及宽高比，角色依然可能会过于靠近相机，进而导致部分场景环境出现剪裁错误。同时，还应注意玩家的武器系统/装备系统是否超出角色的碰撞体范围并导致几何体相交。该问题可通过以下方式加以解决，即在全部场景元素之上渲染武器系统/装备系统，但根据射击位置，此类方案仍有可能产生其他问题。

第三人称相机也有可能产生类似的问题，尽管根据相机定位类型，玩家出现上述问题的几率大大降低。如果相机位置处于完全控制下（如基于约束条件的路径相机），则提出不会出现此类问题（除非游戏对象与相机相交）。通常情况下，可通过相机的碰撞体空间避免此类问题的出现。本章后续部分还将再次涉及这一内容。

距离约束条件

如前所述，针对目标对象的快速加速运动，第三人称相机有时则对此束手无策。当处理此类问题时，一类简单的方案是使用固定距离约束条件（二维或三维环境）以防止相机产生滞后现象。同时，该方案还可确保相机与目标对象之间保持最小距离，但应注意以下问题：首先，当相机瞬移至某一位置并满足相应的距离条件时，应确保相机与场景元素或其他游戏对象之间不会发生碰撞。该问题对于正常的相机移动而言普遍存在且应引起足够的重视。如果新相机位置与几何体或其他对象相交，则应考虑使用其他移动方案。其次，美学需求条件所规定的某些特定的移动方式很可能与距离约束条件产生某种冲突。除此之外，在某些情况下，距离约束条件甚至无法对一些游戏体验操作提供有效的支持。因此，此类功能应通过脚本机制或其他设计方案加以解决。

技术要点

下列内容显示了一类相对简单的距离约束方案：

```
Vec3 const unconstrainedOffset = currentPosition-desiredPosition;
// i.e. back from the desired position towards the
// current position
float const currentDistance =inverseDirection.Magnitude();
float const newDistance = Math::Clamp(kMinDistance,currentDistance, kMaxDistance);
Vec3 const newOffset = unconstrainedOffset.AsNormalized() * newDistance;
currentPosition = desiredPosition + newOffset;
// N.B. It's still necessary to check the validity of
// this movement
```

另一类不同方案则将目标对象速度作用于相交速度之上。若相机距离在最大（或最小）允许值附近波动时，该技术可有效地处理运动的不连续性问题，因而备受推崇。另外，运动过滤器也有助于减少这一类波动问题。

平面约束条件

类似于前述纵向约束条件，某些时候需要确保相机位于平面的一侧，或将其运动轨迹限制在某一平面内。其中，后者相对简单——这里，假设暂不考虑碰撞检测问题。对于任意朝向的平面而言，应确保相机与目标对象之间的相对位置生成正确的方向。除此之外，某些时候应禁止相机与平面之间的距离过于接近，但必要时，相机可快速移至该平面的另一侧。与此相反，当相机位于距平面既定距离之内时，该方案通常易于实现且可视为表面约束中一类最为简单的形式。

技术要点

针对于无限约束平面，确定相机距该平面的距离并非难事，并可有效地防止二者之间的距离过于接近。类似地，相机与平面间的限定操作仅需要目标对象的投影以及平面上的参考点即可。

此处，假设平面采用法线和常量方式加以定义（该常量表示为平面在法线方向上距世界坐标原点之间的距离），一类可能的距离约束方案如下所示：

```
CPlane const plane(normal, constant);
// a plane class may offer projection functions,
// etc. anyway
Vec3 cameraPosition(GetCurrentPosition());
float const planeDistance = Vec3::Dot(plane.GetNormal(),
      cameraPosition) - plane.GetConstant();
if (planeDistance < kMinDistance)
{
  cameraPosition += plane.GetNormal() *(planeDistance - kMinDistance);
  // assumes camera is on the " correct " side of the
  // plane, otherwise check the plane normal direction
  // against the vector from the closest point on the
  // plane to the camera, using a Dot product
}
```

表面约束条件

表面约束条件可视为平面约束的扩展方案，第 7 章曾对此进行了较为详细的阐述。然而，当前表面并未定义相机的绝对位置，以防止相机穿越该表面。

空间约束条件

类似地，空间体也可对相机的移动方式加以限制。针对于此，可执行相关测试以判断点、球体或其他相机碰撞图元是否位于当前空间体内。鉴于相机约束所采用的空间体多为一类简单的凸体形状（如

圆柱体），因而此类测试通常易于实现。考虑到测试所产生的处理器性能开销，普遍采用的做法是将相机视为一点。需要注意的是，可适当地扩展空间体以防止近剪裁面与渲染几何体产生相交。

玩家操控方式

根据游戏或设计风格，大多数游戏支持（或需要）多种不同方式的、基于相机位置和/或方向的玩家操控。通常，针对第三人称相机而言，由于相机系统无法针对场景环境或游戏体验需求的复杂度问题提供有效的解决方案，因而相机的定位多出现于玩家的肩部位置处。多数时候，可放弃该功能项转而由玩家进行操控。然而，作为一类通用处理方案，此处并不推荐使用该方案。在游戏体验过程中，为何令玩家对相机实施操控？其答案在于：玩家具备多种操控途径。

亦即，玩家可不断尝试将相机置于有利位置处，与此同时，玩家并不会中断任何游戏体验活动。另外，鉴于该过程可适当地简化开发过程，因而这对大多数游戏开发人员而言颇具吸引力。一种观点认为，设计人员应具有海纳百川的胸襟，若不考虑其合理性，该观点则充满了崇高的理想主义色彩。此处，该观点意味着，不应强迫玩家对相机进行操控，除非游戏设计方案中对此加以明确要求。

毕竟，玩家的注意力更多地集中于玩家角色的操控之上，而非对相机进行定位和定向。当然，凡事皆有例外，如果游戏设计方案未作出明确规定，此处也可对玩家操控加以禁止。相关示例包括传统的第一人称游戏中的全视角方案、车辆模拟器中的固定相机定位方案等。尽管全视角多出现于第一人称游戏中，但鉴于该方案使得玩家可对猜谜类游戏以及场景环境实施较好的操控，因而也可应用于第三人称相机中。针对于此，有必要将相机操控限制在特定的环境中。例如封闭场景环境，其中，相机的位置需由设计人员加以显式的控制，仅提供有限场景视野的猜谜类游戏等。

仅当玩家角色控制方案相对于角色或屏幕时，相机位置操控方案方为有效。如果控制方案相对于相机，则相机的移动行为将改变控制参考帧并会产生某些负面影响，至少对玩家操控而言的确如此。然而，某些游戏并未对此引起足够的重视。不考虑玩家因素即修改控制参考帧将与良好的游戏体验相悖。考虑到玩家角色的每次移动都将对控制方式进行修改，因而将大大降低游戏的体验感。

另外，保持控制方案的一致性将是一项十分重要的内容。例如，如果第一人称视觉画面中采用了全视角操控方案，则同一控制方案亦适用于其他全视角实例，其中也包括第三人称视觉画面。自然地，该操控一致性特征也可用于玩家操控的其他方面，针对基于控制参考帧的玩家移动而言尤是如此，具体内容可参考第 2 章。

相机位置的控制方案

如前所述，在第三人称游戏中，玩家可对相机位置进行直接操控且此类操作方式十分普遍。下面将分别考察相关操控方案对 2D 和 3D 相机的作用方式。

2D 相机

针对于 2D 游戏，玩家唯一的可控内容即是相机的场景位置（毕竟其投影方式为正交投影）。然而，某些 2D 游戏还支持相应的缩放效果。例如，可沿垂直于 2D 投影方向的轴向模拟跳跃动作（亦即，平行于相机投影方向），并从“上方”进行察看（或“自上如下”）。该过程可通过缩放渲染窗口的尺寸加

以实现（如果采用贴图方式或精灵方式加以绘制）。同时，该操作方式还可实现 3D 效果的近似模拟。需要注意的是，当画面缩小时，应避免相关像素的丢失；当画面放大时，则应避免出现斑驳状图案。若精灵或背景图案采用高于正常分辨率的方式加以存储，并在随后的常规渲染操作中执行缩放或抗锯齿操作，则可在一定程度上缓解该问题，甚至可完全消除此类问题。在维护正交投影时，还可对渲染窗口实施旋转操作。随着现代图形硬件的不断发展，整体屏幕均可视为一个纹理多边形，且可在必要时进行适当的转换操作。

在滚屏游戏中，例如传统的横向卷轴游戏，玩家可预先查看其运动方向。与正常的视见画面相比，某些游戏还支持下蹲或仰视等动作，并适当地移动相机以获取更为广阔的游戏场景画面。当然，此类移动方式依然需要限定于当前游戏场景边界内。通过移动相机位置，自然会改变玩家角色的屏幕相对位置，因而正常状态下，应保持玩家角色的可见性。另外，缩小效果可通过重新调整场景渲染画面加以实现，进而针对游戏机场景内的相对位置向玩家提供较大的透视图。

2D 游戏通常会对相机的移动加以限制，以使场景投影视见内容不会显示既定贴图之外的场景内容，除非游戏体验需求对此加以特别说明。当限定相机位置后，屏幕内的玩家角色位置也会相应地产生变化。

3D 相机

由于可从游戏主角眼部位置对当前场景加以观察，因而第一人称相机通常不允许玩家对相机位置进行操控。尽管某些时候模拟器以及与车辆相关的游戏可视为第一人称游戏，但严格地讲，此类游戏应隶属于第三人称范畴，尽管它们常从玩家角色的眼部视点位置提供相应的场景画面。相对于车辆对象，该类游戏常可根据不同的距离、高度提供不同的视见画面。针对多个相机位置间的循环变化，单一操控方案依然足够，当然，实际操作还应取决于具体的设计方案。另外，相机不同视见内容之间的切换还可通过跳转镜头或插值计算加以实现。针对于保持相同参考帧的位置数据，插值计算可视为一类首选方案。飞行模拟器以及其他角色相关的玩家控制方案，其相机视见画面往往不涉及处于操控状态下的车辆的移动方向，且相机很可能会直接跳转至某一新位置处。

观察相机

某些多玩家游戏则采取类似于调试相机的方式以支持玩家对相机的操控。亦即，相机的移动以及重定向自由度与当前正在进行的游戏体验过程无关。该情形多出现于观察型相机中（即相机并未真正参与到当前游戏进程中），其中，相关角色已从场景中移除。这一类相机亦称作观察相机且通常处于完全自由状态下，抑或伺服于某一特定游戏对象。针对于后者，针对游戏场景内的不同玩家角色、AI 敌方角色以及多个游戏位置，相关操作常包含其间的目标对象快速选取功能。类似地，玩家还可从游戏对象自身角度察看当前场景世界。这将对当前玩家提供某种启示（即从另一角度察看当前游戏场景世界）以及强有力的调试工具，以进一步理解 AI 对象的行为方式——此处，假设 AI 对象包含视线以及其他视觉元素等行为特征。

相机移动确认方案

在第三人称视野中，基于玩家控制的相机移动将根据目标对象对当前相机加以定位。当玩家对相

机进行操控时，若角色仍可移动，则问题将会上升至有效相机位置确定方案这一高度。自然地，相机不应穿越或位于场景几何体外部，但经玩家操控后的期望位置则无此限制要求。令相机位于当前场景环境内其实现手法通常较为直接，但考虑到相机可能被遮挡或最终位置与玩家所操控的期望位置相悖，因而该过程可能会出现问题。当插值或移动方案可通过计算方式获取进而确定相机的可见性时，相机可采取预期移动路径方案。但通常情况下，可向当前场景环境投射光线或设置相应的碰撞体加以实现。当相机遇到静态几何体时，可自动跳过该障碍物（其副作用在于，整体场景皆会发生改变——该情形与跳转镜头类似）或直接选择插值路径。另外，动态几何体（如游戏对象）也可采用统一方式加以处理，抑或调整其渲染属性以防止遮挡或相交现象的发生。

在第三人称相机中，某些游戏并不提供针对相机位置的控制方案，而是设置距目标角色的距离和高度值。相机的玩家相对位置将处于相机系统控制下，进而可消除前述碰撞问题。该操作在赛车类游戏中较为常见，这一类游戏多针对趣味性提供相应的选取方案，而对于实际相机视见画面则较少涉及（如挡泥板）。不同相机视觉画面之间的切换操作可通过跳转镜头或插值计算加以实现。

若相机可围绕玩家角色移动，则会涉及若干潜在问题。这里，针对基于玩家控制的相机移动方案，建议采取以下相关方案：

（1）相机不应移至某一位置且该位置导致玩家角色全部被遮挡。

（2）如果移动投影弧致使相机到达某一位置且该位置不会产生任何遮挡现象，在某些情况下，则可采用相机操控方案。当然，此处问题在于，相机不存在任何义务以实现此类运动方式。然而，该情况可被检测到，且一旦控制过程不再有效，则将自动执行剩余部分的运动轨迹。某种程度上，这也可视为一类推荐方案，该方案将解除相机操控过程中的某些限制条件。必要时，应确保玩家对相机的操控感，这一点十分重要；同时，还应防止玩家角色被长时间遮挡。

（3）当玩家试图将相机移至某一位置，且该位置受限于相关几何体时（亦即，与几何体相交或完全包含于该几何体中），一类常见的方案是将相机靠近当前角色，以获取较好的移动性。进一步讲，如果该“受迫型”移动导致相机与玩家角色相交，则可在垂直方向上移动相机。鉴于相机移动方向已知，因而预测相机与碰撞几何体之间的碰撞状态则变得相对简单。这里，假设相机相对于玩家角色移动。

然而，此处仍需对该过程实施进一步考察。当在当前位置与相机至期望位置的外围方向上投射光线时，则可对潜在碰撞以及遮挡实施有效的检测。光线投射结果可用于调整期望位置。

鉴于通常可将相机碰撞体视为球体，因而光线检测几何体以及目标对象之间应具备足够的空间。

如果相机需要靠近玩家角色以躲避几何体，并于随后保持当前位置和距离直至玩家停止移动当前角色为止。这可以有效地消除角色途径阈值位置时所产生的相机震动现象。此时，随着玩家不断接近相机，应适当地增加相机的高度值。这里，可采取相应的安全检测条件以避免相机位置与玩家角色之间产生相交。在极限情况下，可对相机实施重定位操作（即跳转操作），但需要注意的是，可能的话，应保持控制参考帧的一致性。

位置控制约束条件

通常情况下，玩家应对相机的重定向操作以及移动方案施加某种限制条件。这可有效地防止相机位于当前场景环境之外或与几何体（无论静态或动态）之间的距离过于接近，进而引发相机的近剪裁面处于相交状态。除此之外，某些方向还将会导致数学运算产生问题，例如当相机的前向轴平行于世

界 Up 坐标轴时。这将使得玩家失去方向感并生成围绕该轴的未知旋转操作。玩家对相机的移动方式加以直接干预可导致相机自身的重定向操作，该方式常使得玩家游戏体验画面显得过于混乱。

相机的玩家相对位置常受到角度的限制，其中，该角度由玩家方向向量与角色和相机之间的向量构成。简单地讲，相对于角色的相机多会受到相关条件的约束，以使其位于玩家角色的后方。因此，玩家有必要对相机的运动行为加以阻止，以防止相机朝向当前角色。当然，在某些情况下，该决策方案可能过于武断，进而使玩家丧失某些功能项。

当基于玩家的相机操控结束后，大多数第三人称相机通常会返回其默认位置，该位置通常具有一定高度且与玩家角色方向保持一致。除了与相机重定位操作相关的约束条件之外，还有必要进一步限制相机的重定向速度。NASA 曾对相机旋转速度与用户舒适度之间的关系进行了相关研究（如[Ellis99]），研究表明，相机的移动应保持平滑、缓慢状态，以避免玩家失去方向感。

相机位置控制方案

关于相机与另一游戏对象之间的相对位置确定方案，此处存在两种对应方案，即相对于角色的定位方案以及相对于世界空间的定位方案。

相对于角色的定位方案

该方案通常与第三人称相机相关，但在第一人称游戏中，相机有时也可相对于角色进行有限的移动（如模拟蜷伏动作）。这里，玩家可围绕目标对象（或视见位置，二者应具备同一含义）实施旋转操作，进而对相机的位置加以控制。同时，还可对仰角（或者世界 Up 轴偏移量）以及旋转半径进行适当的调整。另外，相机移动时的弧度轨迹无须限制为球体形状，多数时候，椭球体则更为适宜。

若玩家可对相机的位置加以控制并生成正确的角色视见画面内容，且同时可对游戏体验选择方案加以限定或影响玩家的控制状态，则该类相机位置操控可视为一类相对完善的处理方案。

这里，可向相机移动以及相机距离范围添加相关约束条件，进而避免潜在问题的出现。这将生成基于可能位置的柱状带，而非球体。除此之外，还可在角色（朝向）后方将围绕当前角色的旋转操作限制在某一小角度范围内。

针对于相机选取方案，可在相对于玩家角色的不同距离、高度处提供多个相机位置数据。相关位置可作为离散相机选取方案或为进一步的玩家操控打下良好基础。

某些第三人称游戏则支持半自动相机移动方案，并有可能将玩家角色移至某一危险位置处进而显示对应的场景画面。相关示例包括悬崖处的眺望动作、墙角处的察看动作等。尽管上述动作处于严格控制下，但均可视为一类角色相对位置的控制方案。若此类场景可基于预置脚本机制加以显式控制，则可提供多种方式加以实现，尽管此类操作多根据玩家需求得以实现并出现于特定的时刻。

相对于世界空间的定位方案

相对于世界空间的操控方案多与全视角或调试相机有关。此时，相机方向与玩家角色的方向无关。然而，在某些情况下，相机位置依然与玩家角色保持着千丝万缕的联系。当显示伪 2D 场景画面时，可采取此类定位方案。调试相机通常不与其他游戏对象发生任何关系，甚至是玩家，并可视为一类真正意义上的“相对于世界空间”。某些时候，获取较开阔的视觉画面将十分有用，进而可辅助玩家有效地预测下一步动作。例如某些即时战略类游戏、游动类游戏、3D 游戏等。即使这一类相机的操控环境相对轻

松，但某些“正常的”约束条件依然不可或缺，这将有效地防止渲染过程中所产生的某些负面效果。

相机方向控制

基于相机方向的玩家操控亦称作全视角操控（至少在第一人称相机中应是如此）。相反，2D 游戏通常不提供此项功能（2D 游戏多采用正交投影），但却支持全场景的旋转操作（针对某些伪 3D 游戏而言尤是如此，如等轴测投影）或者使用大量的既定方向（如等轴测游戏）。同时，某些 2D 游戏还可对游戏场景实现“摇动”拍摄这一类效果。上述操作可呈现为全自由态或在玩家角色附近处限制为较小的移动量。

全视角可视为第一人称相机中的重要组成部分，并可用于辅助玩家的瞄准操作，以及缓解此类游戏中有限视域范围所产生的相关问题。

第一人称相机

在第一人称相机中，相机的方向控制可根据玩家的移动状态实施单键操控。该功能的主要目的在于实现一类“盘旋俯冲”动作，其中，玩家可围绕目标对象（或位置）进行相应的旋转操作，且依然维持相机与目标对象之间的朝向关系。这可通过推动摇杆（盘旋或侧翻）以及反方向（水平方向）的全视角摇杆（转向操控）加以实现。在某些时候，该过程还可用于模拟鼠标操作。需要注意的是，水平翻转方向应与水平移动方向（盘旋）相反。

同时，也应对全视角操作采取一定的限制，以避免玩家的视见方向平行于世界 Up 轴。相对于玩家角色，当相机到达旋转极限时，应对此添加需要的阻尼效果。类似地，任何源于控制器的方向噪声都应采取适当的过滤机制，以避免相机产生不必要的抖动现象。针对于自身恢复型（中心定位）摇杆，视觉画面将与摇杆位置实现良好的匹配，亦即，当摇杆释放时，画面将自动复位。然而，应尽量使用过滤后的输入数据且通过控制器定义目标位置或控制相机的加速状态，而非直接映射操作。当控制器的灵敏度产生较大变化时尤是如此。

当对相机方向进行调整时，除了某些飞行模拟器之外，尤其是太空飞船模拟器，倾斜旋转操作往往不予支持。当飞行器采取 6 个自由度时，传统意义上的“上方”概念则稍显模糊。然而，该过程依然需要防止出现万向节死锁这一现象。

全自由度操控方案通常假设玩家视见内容与玩家角色的水平方向同步执行。这也意味着，水平方向上的全视角操作等同于玩家角色的转向操作。然而，事实并非如此简单。针对于此，可单独使用全视角操作并用于模拟步行或跑动时躯干的移动状态。当采用此方案时，躯干的重定向旋转量将限制在 90° 范围内。针对于高级玩家而言，该功能将十分有效；而对于入门级玩家，基于任意方向的角色移动模式常会令玩家失去方向感。如果全视角定向方案支持恢复型（中心定位）相机，当释放控制设备时，则可大大降低上述不一致性所产生的负面效果。

第三人称相机

通常情况下，方向变化可通过以下方式获取，即围绕当前玩家移动相机且将视见位置置于玩家角

色上方。因此，相机的移动将被限制在虚拟球体或圆柱体的表面之上。某些时候，相机的重定向操作也可独立于玩家或相机的移动。此处，需要对相机的重定向操作加以限制，并针对玩家角色维护其屏幕位置，且当相机控制被释放时，相机应可自动恢复其原有画面。当玩家在游戏过程中执行瞄准操作时，玩家角色部分内容应可位于屏幕之外，且呈现为半透明状态以对敌方角色进行准确的跟踪。为了实现有效的瞄准操作，有必要将相机置于玩家角色后方位置处。

在第三人称相机中，鉴于相机难以沿武器视线方向进行察看，因而其瞄准操作通常难于实现。该操作包含了丰富的内容，第 8 章曾对此内容进行了详细的讨论。另外，在某些示例中，传统的“跟踪型”相机可围绕玩家调整其视见画面内容，且无须对相机执行重定位操作。针对角色武器系统的绘制，某些游戏则在第一人称相机与第三人称相机之间提供了一类折中方案。其中，相机靠近于玩家角色的肩部位置。与第三人称相机相比（该类型相机与玩家之间保持一定距离），全视角相机则可实现更为紧凑的控制操作。当采用此模式时，可选择性地关闭移动功能。

当选择全视角方案围绕玩家移动相机时，应确保角色不会对视觉画面产生遮挡。这里，可在当前角色上方进行察看，或者将相机置于远距离位置处（如果切实可行的话），以防止角色的屏幕尺寸过于突兀。除此之外，玩家角色还可呈现为半透明状态，且相机不可与当前角色出现相交，除非该角色完全淡出当前画面。

某些时候，相机需要在敌方角色和己方角色之间取景，并允许在角色分组间实施重定向操作。无论如何，相机都需要聚焦于场景中的游戏对象或相关位置，因而有时需要采用屏幕图像元素以辅助玩家的选取工作。另外，还可通过标线图案辅助玩家的瞄准操作，该图案不会对场景画面产生任何遮挡。

这里，标线多采用十字线或类似图案并以二维形式加以显示，当与透明度结合使用时，可进一步降低对游戏元素的遮挡效果。少数情况下，十字标线则通过三维模型加以绘制，由于此类标线着重强调场景世界的三维特征，因而该技术可大大增加标线的方向感。当玩家角色配备了某些虚幻装配后，如头盔，鉴于这一类装备通常会包含某些特定的功能项，因而需要通过图像方式凸显其迥异风格。针对于渲染操作，标线图案可视为一类二维元素。同时，还可根据距目标对象的距离适当地调整标线图案的尺寸数据。需要注意的是，即使目标对象距相机较远，也应保证标线图案的尺寸足够大，并确保玩家对其可辨。类似地，标线图案的最大尺寸同样不可或缺。

相机的自动定位和定向方案

在某些情况下，相机系统应可对当前相机实施重定位/重定向操作且无须玩家直接干预。但出现此类情形时，应禁止玩家与相机之间的交互操作。其中，较为常见的示例是电影视频序列，多数游戏体验均会包含此类元素。

基于玩家控制器的反馈

玩家可选取相应的相机位置和/或方向数据，并自动应用于相机系统中。当相机的可控量有限时，则多采用此类方案，如车辆模拟器。这里，可制作与预定义相机位置和行为相关的“调色板”模型以简化玩家的选取操作，其中，可以固定顺序执行循环选取操作，或直接将相机位置映射至控制器的特

定按键之上。

有时，上述选取方案可能会出现即时变化，例如，在飞行模拟器中，飞行器的状态将导致视见画面发生变化。座舱四周的视觉内容通常会在短时间内产生变化。除此之外，相关内容还包括一段时间的位置和/或飞行的插值计算。车辆模拟器在不同距离范围内将生成不同的视见内容，相机系统可对其实施平滑的插值计算。

基于游戏体验需求的反馈

相机的自动定位存在多种原因，在第三人称游戏中，应确保特定的游戏体验对象位于玩家视野中。除了取景问题之外，如果相机的重定位操作独立于玩家控制操作，应确保新位置易于理解且控制参考帧的变化也应具有一定的意义。

在某些场合，增强玩家相机的操控能力将有助于实现良好的游戏体验感。针对狭隘的场景地形，第一人称相机的表现能力往往难以令人满意。因此，玩家难以理解其场景位置与环境特征之间的关系，例如需实现跨越动作的沟壑地形。又如，当玩家靠近悬崖边缘时，可对相机的仰角加以微调；或者，玩家在跳跃过程中也可适当地调整其仰角（进而可对着陆位置实施良好的预判），这将大大增强玩家的游戏体验感且不会对整体画面产生任何不良影响。针对后者，仰角量可根据滞空时间、与目标位置的接近程度，甚至是玩家输入数据进行适当的调整。

在第一人称游戏中，当玩家沿斜面上行或下行时，若相机方向依然与水平面保持平行，则其对场景的观察能力将变得越发困难。一类方法是可自动调整相机的上、下仰角，并保证相机的静止位置（即未受到玩家全视角操作时的位置）始终平行于（或接近于）水平面。当玩家沿斜面上仰或下调其视角时，仰角值应根据玩家方向与斜面之间的夹角按比例加以调节，当夹角值为 90° 时，仰角值则为 0° 。

在第三人称游戏中，游戏中的部分内容可采用伪 2D 方式加以展现。针对于此，相机可以平行于世界轴的方式加以定位，甚至固定于某一样条路径上。多数时候，相机通常不与玩家的速度发生直接关联，否则，将使得相机缺乏应有的自然感；另外，玩家的轻微移动即可导致相机产生大幅度的运动。

如前所述，某些情况下，玩家需要适当地移动相机并可对某些危险的环境区域实施察看。有时，可简单地提供一类定向（或定位）相机以生成新的视点；抑或将世界偏移相机与玩家角色结合使用（如倚靠于墙角这一类动作），当玩家躲避敌方角色的火力或侦察行为时，可有效地缓解玩家对相机的操控需求。

调试相机的操控方式

相机行为的调试和确认过程可视为一项复杂任务，且相机通常缺乏相应的定位和定向观察能力，这一点与实际相机不可同日而语。因此，此类相机应具备一定的定位功能并可从某一视点处对当前场景世界进行观察。另外，相机还应具备屏幕捕捉能力，且相关游戏状态的取景工作应符合一定的美学要求。

除去常规游戏相机之外，相机系统还应提供相关机制以实现这一类辅助型相机，以简化相应的调试过程。用户需要对此类调试型相机进行操控，且不应对其他游戏对象产生任何影响，其中也包括其他类型的相机。针对于此，前述内容曾提及了观察型相机。因此，有必要对场景渲染代码中的转换内

容加以覆写，但不应改变输入数据与其他相机之间的接收关系。这里，多调试模式可视为一类有效的解决方案，其中，某些调试策略将对输入数据采取具有针对性的传递操作。此时，调试相机的移动过程不会对当前玩家角色产生任何影响。如果系统支持控制器操作，抑或输入数据可通过其他外部机制获得（例如 PC 与开发工具包之间的控制台环境，而非常规的游戏控制器），上述过程则毫无必要。当采用回放功能对 bug 进行复制或实施功能检测时，调试相机则可自由移动并对当前游戏场景实施有效的观测，该功能极具应用价值。

一般地，调试型相机通过偏航角和仰角调整其方向状态以及沿其当前前视方向上的往复移动。另外，相机还应支持侧向倾斜旋转操作，该操作垂直于前视方向且原 Up 向量保持不变。除此之外，还可有选择性地对相机实施加速操控，以便于相机沿同一方向进行快速移动。一种移动和方向控制的替代方案是锁定相机，并使其移动与游戏对象相关，或与视见方向上的某一既定距离位置相关。上述技术大大简化了相机预测位置及方向的测试过程，同时，还可确保用户更好地对游戏对象行为加以观测。

另一类调试技术还可动态地降低游戏的执行速度，甚至可达到每次游戏更新操作中基于单步执行的断点这一程度。这可通过调整游戏对象的时间间隔量加以实现。当然，除去调试型相机之外，该方案也适用于游戏相机。因此，调试型相机可以正常的速度移动并执行相应的重定向操作，即使游戏逻辑的运行较慢，玩家依然可对相机的位置（和方向）进行交互式控制。进一步讲，独立于调试相机的、游戏逻辑的单步运行方案则更胜一筹。理想情况下，相机还应具备将当前游戏恢复至前次状态的能力，当然，该过程实现起来较为复杂。

相机碰撞

相机的碰撞问题与游戏相机的移动方案具有同等重要的地位，相关内容不仅涉及碰撞检测问题，更为重要的是，还应针对碰撞预测以及碰撞躲避提供相应的处理方案。注意，碰撞躲避可能会产生遮挡问题，且这一类问题的解决方案有时会忽略相机碰撞系统。另外，还应对相机碰撞的必要性进行相应的考察。例如，在非交互式电影视频序列中，鉴于可对相机的位置加以显式操控，因而通常不需要检测相机的碰撞状态。该情况也会出现于其他游戏环境中，其中，相机位置通过预置方式加以定义，抑或可确保相机实现自由移动（如位于场景环境外部）。需要说明的是，碰撞检测常会产生大量的处理器开销，该问题可通过预计算数据或脚本游戏对象加以解决，如反重力设备。碰撞躲避也可视为漫游形式之一，第 8 章曾对此进行了详细的讨论。

当然，可完全忽略相机的碰撞操作，并使得相机可穿越游戏对象或几何体，该操作并非不可接受（如调试型相机），但此类行为将会产生某些视觉副作用以及其他游戏体验问题。

相机碰撞的重要性

为了有效地避免相关问题的出现，相机的碰撞检测可视为一类预测机制，而不仅仅是相机与游戏对象或封闭空间环境之间的相交计算。下列内容显示了源于相机碰撞中的一类问题：

☑ 碰撞操作阻止相机到达其期望位置。因此，如果不存在相关的安全检测机制，则相机应滞后于目标对象。

- ☑ 不连续移动将产生碰撞现象。碰撞检测使得相机移动无法按期望意图执行，因而相机移动将变得十分不规则，进而出现不连续现象。当碰撞几何体包含大量细节内容时通常会产生此类情形（本章后续内容将对碰撞几何体设计问题进行详细讨论）。
- ☑ 目标对象的遮挡效果。如果对象或几何体与相机间的碰撞位于相机以及目标对象之间，则该目标对象处于遮挡状态。考虑到前述相机移动所产生的相关问题，因而该情形屡见不鲜。
- ☑ 几何体与相机的近剪裁面相交。在相机接近对象或几何体的过程中，如果相机碰撞体的空间范围小于视锥近剪裁面的距离，渲染对象与近剪裁面相交的最终结果将导致渲染操作产生某些负面效果。针对于此，可适当地增加相机或对象的碰撞体以消除此类问题。第一人称相机也会出现这一类问题，其常见处理方案是调整问题对象的碰撞几何体，而非玩家角色的碰撞体空间，此操作向玩家提供了一类可选方案。

因此，碰撞与相机之间往往具有紧密的关联。一方面，如果设计人员未对某些可控环境加以说明，则相机不应位于游戏机场景环境的外部空间。同时，相机不应脱离于其目标对象而移动。除此之外，如果相机过于接近对象或环境元素，则应避免相机发生穿越现象；否则，当前环境或相关游戏对象将产生错误的渲染结果。

然而，某些游戏对此类问题并未引起足够的重视，抑或无法提供高效的处理方案。或许，紧张的开发周期使得开发方案无暇顾及此类问题。无论如何，上述问题将极大地降低游戏的观感质量，且难以得到玩家或旁观者的肯定。不可否认，这一类副作用难以令人接受，高质量的游戏不应包含此类错误。

处理（或避免）环境或对象间的碰撞问题并非易事，其时间消耗亦不容小视。这里，当检测到碰撞问题后，相机应如何移动？该问题通常取决于相机相对于玩家角色之间的期望位置。考虑到当前环境将阻止相机到达其期望位置，那么，相机是否可尝试各向移动？在某些情况下，鉴于相机高于当前角色位置，因而无须执行上述移动操作。然而，根据具体的操作方案，上述操作依然会产生问题。考虑到数学计算中的问题，通常情况下，应避免相机与玩家角色垂直对齐；相对于相机而言，左移或右移将变为围绕垂直轴的旋转操作，这并非是期望中的结果。

另外，碰撞检测还取决于游戏场景内的物理表现方式。多数时候，出于碰撞考量，相机可视为球体或圆柱体形状，而球状碰撞体则往往更为高效。

通过预置相机位置（无论固定于原处或沿既定路径移动）或预测相机及其支配对象的未来位置，可有效地消除与碰撞相关的问题。实际上，当场景几何体相对复杂时，预测型相机往往可视为一类有效的方案。

碰撞检测的必要性

在实际操作过程中，相机碰撞检测并非必需。例如，当玩家（或相机设计人员）显式地指定某一位置，且该位置不会与场景几何体或游戏对象产生任何碰撞时，该情形多出现于非交互式视频序列中，但也适用于正常的游戏体验环境中。

预定义相机虽然高效，但多与游戏类型紧密相关。由于相机移动确定方案固定不变，因而相机在某种程度上呈现为禁止状态。另外，针对复杂环境的游戏体验需求，或处于变化中的游戏体验交互过程，获取一条有效的移动路径或相关处理方案通常比较困难。实际上，在特定的游戏区域中，往往需要使用多种方案。然而，考虑到相机碰撞检测所涉及的处理器开销问题，此类方案依然较具吸引力。

需要注意的是，相机定位约束条件通常不会导致相机行为的整体预测这一现象。例如，针对空间体内相机的不同移动方式，空间体约束条件均提供了相应的空间范围。关于相机定位方案，第 7 章曾对此进行了较为详细的讨论；另外，读者还可参考本章开始部分以获取与移动约束条件相关的更多内容。

需要强调的是，某些潜在碰撞可安全地排除于测试之外。如果问题几何体仅仅是附加于当前环境之上的一类美学元素（如网状电线或细长电线），则相机依然可穿越某些狭长的几何体。这里，唯一需要注意的一点是，几何体不应与近剪裁面相交。该问题可通过以下方式加以解决，即在相机的移动过程中，可对问题几何体实施“淡出”效果（如果该几何体独立于当前场景环境进行创建，则此操作易于实现）；或者，当产生相交时，可不对问题对象实施渲染操作（当然，这将导致该对象不断地“跳入”或“跳出”当前视觉画面）。

碰撞确定方案

针对于相机与场景世界或对象几何体之间的碰撞决策方案，存在多种方法对此加以处理。首先，读者需要考察是否有必要对相机执行碰撞处理；另外，若将相机视为一类对象且可与虚拟世界产生碰撞，其优势在于何处？

通常，第一人称相机无须执行任何碰撞检测，该任务多由玩家角色加以隐式处理。然而，应确保玩家角色碰撞体具有足够的空间，进而防止第一人称相机的近剪裁面与几何体相交。然而，第三人称相机则包含诸多问题，碰撞躲避仅是其中之一。因此，此处讨论多集中于第三人称相机。既然如此，针对于涉及巨大处理器开销的碰撞计算而言，其意义何在？首先，相关方案可减少或消除近剪裁面与几何体之间的相交问题（该原因已然足够）。其次，由于相机在玩家的作用下可沿碰撞几何体表面滑行，因而可大大降低对相机脚本机制的需求。最后，碰撞处理过程可使相机位于当前场景环境中——这也可视为一项附加条款。

对象碰撞

多数时候，相机与游戏对象之间的碰撞计算并非易事。因此，相机需进行适当定位以避免出现碰撞现象；否则，应尝试考察下列内容。其中，近剪裁面相交问题可视为重要考察对象，然而，游戏对象也可能阻碍相机的移动，或导致相机移动出现显著变化，以避免游戏所导致的遮挡现象。

环境碰撞

若计划对碰撞现象采取某种躲避措施，则应考察相机与对象间的接近方式。多数游戏引擎多采用两类彼此相关的数据结构以表现游戏场景环境：首先，存在一类用于渲染操作的数据，例如三角形网格或高阶对象表面，这一类数据多与可见性预计算以及其他空间划分信息结合使用。这里，可将其称之为渲染网格或渲染几何体。其次，低复杂度的对象表面表现方式多用于碰撞检测（如碰撞网格或碰撞几何体），当获取碰撞信息时，其低密度特点将大大降低处理器开销。除此之外，还可对碰撞网格进行适度调整，亦即，通过提供相应的平滑边，以便于玩家在场景中实现移动漫游。碰撞网格通常采取远离渲染网格的方式进行定位，以使碰撞出现于渲染网格相交之前。需要注意的是，应避免碰撞呈现

为较明显的“浮动”状态。这里，应保持低复杂度外观以及相关美学需求条件之间的平衡性，针对于此，有时则需要采取某种折中方案对此加以处理。另外，还可对渲染操作以及碰撞网格进行细分，并将其视为彼此独立的两部分，或根据各类不同的因素（如与相交的接近程度）实施动态调整。通过限制需测试的图元数量，该方案可有效地降低处理器计算开销。类似地，网格复杂度也可随时发生变化，即细节层次（LOD）。关于碰撞问题，其详细讨论则超出了本书的范围，读者可参考[Ericson05]以获取更多内容。

碰撞图元

出于碰撞角度考量，如何在游戏场景中描述相机？此处，应防止相机过于接近几何体，以避免出现视锥体剪裁这一现象。通常，尽管相机实际上定义为一点，但从碰撞角度出发，可将其视为一个椭球体。另外，光线投射机制也可视为一类备选方案，实际上，高效的光线投射方案为相机漫游以及遮挡消除提供了坚实的理论基础。因此，针对相机碰撞问题，需要在相机碰撞球体与碰撞网格之间执行测试。另外，并非全部碰撞几何体均适用于相机碰撞检测，因而需要根据多边形网格中的表面特性实施碰撞过滤，针对于其他游戏对象，这也使得相机可穿越更为坚实的几何体。

在极度复杂的场景环境中，可采用较为传统的方式避免碰撞的发生，例如，采用既定路径以防止相机移至几何体内部。针对静态几何体，这已然足够，但该方案并不适用于游戏对象。

这里，场景的表现形式可包含以下两部分内容：首先，静态几何体构成了场景中固定不变的部分内容；其次是动态几何体，这一类几何体由玩家及其关联对象，或者场景中的其他对象构成。另外，动态几何体可用于设置某些环境特征，这一类特征可在游戏体验过程中发生变化，例如，桥梁可在游戏某一时刻被摧毁。同时，动态几何体还可能包含潜在的移动趋势，针对相机系统而言，将导致问题的出现。如果动态几何体在不同状态间移动，则有必要针对各状态确定不同的相机解决方案。考虑到移动对象可能会阻止相机的定位，抑或在各状态下采用不同方式对目标对象加以遮挡，因而上述方案的必要性不言而喻。

碰撞几何体设计

针对相机碰撞而言，采用与通用对象碰撞相同的碰撞网格并无不可，但此类网格可能无法满足当前要求。例如，碰撞网格通常经过精心打造，以匹配或接近于场景渲染几何体。例如，武器装备的发射物或类似对象击中碰撞网格时，针对于渲染几何体，其过程应符合相应的外观要求。出于此目的，该方案无可厚非，但对于碰撞表面附近处的平滑相机移动而言，此方案并不适宜。针对相机碰撞问题，此处将使用一类相对简单的平面，以使相机沿对象表面平滑地移动。同时，还应将直角对象改造为圆角对象并以此对同一类型的移动提供良好的支持。

当与相机碰撞表面相关的需求定义完毕后，如何实现这一类碰撞表面？是否还存在其他约束条件以影响相机碰撞网格的设计方案？碰撞过程的表现方式往往取决于游戏类型、表现风格以及环境类型。针对此例，下面将着重考察第三人称相机，其中，玩家计算漫游于某一封闭场景环境。当对上述场景环境进行设计时，下列内容应予以考虑。

避免紧凑型封闭空间

环境空间越狭小，相机的定位就愈加局促，同时，针对于游戏体验的取景操作，将大大降低相机的定位可选余地。除此之外，玩家计算将更容易与相机相交，且相机按原定路线的行进几率也将大大降低。更为复杂的是，当相机再次返回至封闭空间内时，需对玩家附近位置进行搜索，以实现最终的有效定位。

向相机提供足够的空间

考虑到多数第三人称相机均受到玩家角色的支配，因而应确保相机拥有足够的空间，以实现相机与玩家角色之间的距离定位。如果当前场景环境无法实现第三人称相机的有效取景模式，则应尝试特定的相机方案，以对当前游戏体验进行较好的描述。在某些情况下，相机需要定位于渲染几何体的外部空间内，因此，应适当地添加某些“辅助型”渲染几何体，以防止某些游戏场景元素生成不适宜的画面内容（如场景环境的背面图像）。

陡峭的阶梯

当目标对象从阶梯上行进时，常会出现相机的遮挡现象。这里，将阶梯调整为圆角对象将有助于问题的解决。当然，具体操作过程还取决于玩家角色相对于阶梯尺寸的高度值。

突兀地形

针对于此类地形，玩家在下坠过程中可能还包含后退动作，该过程将导致相机产生问题。如果玩家位于此类地形下方且相机仍位于地形表面上方，则相机最终将被地表隔离。一类解决方案是，令相机可穿越当前对象表面，尽管该方式并不予以推荐且常会产生不连续的移动状态。除此之外，还可生成相关路径，并在此类地形上移动相机。这将导致相机产生 180° 的旋转操作，并使得后续控制参考帧发生变化。

低矮的天花板

再次强调，如果相机位置未采用预置方案，例如样条路径或类似的解决方案（相关内容可参考第 7 章），则应尽量避免使用天花板这一类对象。

入口

针对于全视角型第三人称相机，入口的漫游操作通常难以实现。下列内容列举了一些常见问题。

首先，针对于位于玩家角色和相机之间的一扇门，在其关闭过程中，玩家可能会趁机快速穿行（这

将导致相机穿越几何体或快速移动至某一新位置）。针对于此，可迫使门对象保持开启状态，直至相机穿行动作完成为止。另外，当门对象徐徐关闭时，相机不应再次返回并穿越门对象。

其次，玩家穿越入口时，可能会导致相机出现滞后现象（例如，非定向相机仅维持与玩家角色之间的既定距离）。通常情况下，入口越狭窄，则相机对漫游的要求就越高。针对于此，适当地增加门对象的尺寸将有助于缓解此类问题。然而，如果相机系统意识到当前相机过于接近门对象，则可向该相机施加作用力并将其置于门对象的中心线位置处。此处，当相机沿门轴向方向移动时（即垂直于门对象的开启方向），则相机很可能获取较好的视觉画面且与玩家移动无关。通常情况下，仅当入口位于相机与目标对象之间时，方可对相机施加外力。当玩家角色位于入口内侧且执行简单的旋转操作时，也会产生相关问题。此处，若相机位于角色后方位置处，则相机将以弧度轨迹移动，这将与周边几何体相交。同一问题也会出现于墙面以及其他类似结构中，而入口则通常更为狭窄，针对于相机的纵向位置尤是如此。因此，应适当地移动相机并使其跨越门槛，以防止相机与玩家角色相交。

采用简单的碰撞网格

碰撞网越简单，则相机就越容易在其表面上滑行。相机的全程移动应呈现为平滑的状态，而简单的碰撞表面则有助于实现这一行为。另外，简单的平滑表面则有助于相机的停滞几率。

针对相机的独立碰撞网格

针对相机而言，独立碰撞网格使得相关工作得到了大大的简化。除此之外，简单的对象表面还可降低相应的存储需求。

圆角对象

针对于有限的空间范围以及门廊等场景，圆角对象十分有效并可辅助相机实现入口的穿越动作。此处，圆角对象则有助于相机沿碰撞表面实现滑行操作。

碰撞决定方案

一旦检测到发生碰撞，则应即刻确定该问题的处理方式。除了相机移动行为以外，碰撞现象还可能源于其他因素（如对象移至相机内部），但此处的讨论范围则仅限于前者。另外，某些潜在的选择方案也可视为解决碰撞问题的有效方法。

保持静止

此类方案最为简单，该方案假定相机已定位完毕且不会与其他对象或场景环境元素相交。然而，玩家的移动不受任何约束且可自由驶离于当前相机，这将增加相机返回至期望位置时的难度。最终，该过程将导致执行某一安全检测条件，其中，相机需要迅速移至新位置处，并满足相对于目标对象的

期望位置；另外，此过程还需要处理相关遮挡问题。该方案的另一个问题是，在狭窄空间内，相机无法驶离于当前玩家角色，因而将再次出现相交问题。

部分移动

该方案是指相机在短距离范围内（与前次定义的碰撞距离呈某种比例关系）根据期望方向进行移动。如果该移动过程依然失效，则相机应保持于当前位置处。当然，可适当地调整期望位置以尝试各类移动方案。需要注意的是，处理器计算开销在一定程度上限制了操作的循环迭代次数。

在该方案中，相机在其期望方向上的移动距离并无太多限制且不会出现碰撞现象。大多数碰撞系统均支持简单的光线投射测试。

移动变换

如果相机的潜在移动方案与环境几何体之间产生碰撞，一种可能方法则是调整相机的移动方向，该过程可视为相机的漫游形式之一。针对于自由态相机，若其关注点仅限于维护基于玩家角色的距离和高度值，则可通过检测相机与玩家角色之间的相对位置以确定各种变化位置数据。这里，相关位置的确定应使得玩家角色获取适宜的视觉画面，同时还应满足相机当前位置与期望位置之间的移动可能性。当然，这一类定位方案可能会导致某些问题的出现，且在很大程度上取决于期望位置确定方案自身所包含的本质特征。不仅如此，碰撞几何体很可能会阻止相机与新增期望位置之间的直接移动。针对于此，某些漫游方式则会围绕障碍物生成一条移动路径，但在较为复杂的场景环境中，该漫游方案往往难于实现。如该情形不可避免，可尝试采用另一类处理方案，即镜头跳转。

镜头跳转

当碰撞几何体阻止相机的移动时，一类常见的解决方案是重新计算相机与目标对象之间的相对位置，并将相机即时定位于该位置处。这也可视为安全检测的一种具体操作形式，具体内容可参考第 8 章中的相关内容。考虑到该方案可能导致玩家失去方向感，因而应对此谨慎使用。而且，某些游戏还会根据该机制令相机直接穿越障碍物，但这将违背 30° 角规则。需要强调的一点是，针对于相机的瞬移行为，应确保新位置的有效性。

禁用碰撞检测

初看之下，该标题令人稍感意外，但针对相机而言，确实存在禁用碰撞检测这一情形（以及碰撞躲避机制）。

相关示例包括第一人称相机和第三人称相机之间的切换操作。如果平滑的切换操作不可或缺（与之相反的例子则是跳转操作），则根据定义，相机将会穿越基于主角色的几何体。此处，当相机处于近距离状态下时，将角色渲染为透明对象并结合相机的快速移动即可有效地避免近剪裁面几何体相机问

题。其中，可根据相机距角色之间的距离，或根据切换过程中的时间间隔动态地调整角色的透明度。然而，若开始阶段相机距玩家角色的距离处于变化状态，则此类方案通常难以调试。另一种处理方案则是使用全屏过滤效果（如实施全视口混合操作）。

禁用碰撞检测的另一类情形是，相机有时会定位于场景环境的外部空间中，并据此位置执行切入/切出操作。多数（2D 或 3D）横向卷轴游戏均会从场景外部提供相应的视觉画面。某些时候，相机需要扫掠过几何体且无须执行碰撞检测。此时，相机往往通过镜头跳转以穿越于几何体中，进而可避免出现几何体相交现象。

进一步讲，某些游戏采用了传统的、基于图层的卡通动画技术，图层接近相机的近剪裁面时将褪至透明状态，鉴于不再对该图层实施渲染操作，进而可有效地避免出现某些突发性画面。如果该图层未被渲染，则当玩家角色在距离边界值附近移动时，此方案则可有效地避免某些问题的出现。

另外，鉴于既定路径已对碰撞问题实施了有效处理，因而路径相机以及其他移动约束型相机通常也不会采取碰撞检测机制。无论如何，如果希望完全消除近剪裁面所引起的相交问题，静态几何体或移动对象依然需要相机执行一定程度上的碰撞检测计算。针对于此，一类较为简单的方法是，一旦检测到潜在的碰撞问题，则令相机沿其路径继续前移，抑或限制当前相机的移动行为，直至路径期望位置处的潜在碰撞消除。针对于后者，可将相机瞬移至期望位置处，但这将会导致画面出现不连续现象。因此，应尽量避免使用此类瞬移操作。另一类常见的情形是，相机不对碰撞问题进行任何处理，此类情形多见于针对相机的临时遮挡对象，其中，相关游戏体验元素或谜题等待进一步处理或处于未激活状态。一旦谜题求解完毕，对应的遮挡对象亦随之消失（如门缓缓开启），进而相机可沿其路径继续前行。

避免相机碰撞

为了有效地防止相机碰撞问题的出现，可对碰撞时刻进行相关预测，并在事件发生之前对相机的移动方式加以调整。根据当前（或期望）相机位置，常见方案包括光线投射以及空间体投射技术。当然，该过程也包含先期的玩家移动行为预测。源于周边环境的数据信息（如光线投射结果）将对相机与对象表面之间的、相机的期望移动行为产生显著的影响。针对于相机在复杂常见环境中的漫游技术，第 8 章曾对此进行了较为详细的介绍。

某些时候，碰撞躲避机制可视为相机移动方案的内在固有特征。样条或其他路径方案可在一定程度上替代碰撞检测机制，尽管相机的自由度将会受到诸多限制。大多数路径方案经由设计人员对相关路径进行预置（预定义），并根据游戏体验需求以及玩家定位确定最终的相机移动行为。与动态几何体（以及游戏对象）相比，这一类方案可有效地避免相机的碰撞问题。理想状态下，相机移动行为应根据场景几何体的复杂度加以严格限定，抑或确保不与相机相交。

针对相机移动，还可采取动态方式产生相关路径，这与 AI 路径搜索机制具有某些相似之处，但会包含额外的与相机位置与方向相关的美学限定条件。关于路径生成操作，读者应具备与场景空间体及其内存开销相关的某些背景知识。当然，此类方案并不涉及游戏对象。

另一类潜在的解决方案则对游戏对象碰撞问题不予考虑。此时，与游戏相机相交或与其十分接近的游戏对象，可对相关渲染操作进行适当调整（例如，通过 Alpha 混合技术实现淡出效果）。与相机和

游戏对象之间的迂回移动相比，该方案通常略胜一筹。其原因在于，游戏对象的移动方向往往处于未知状态下，其变化也异常迅速——这使得相机亦处于高速移动状态且毫无规律可言。

针对于第三人称相机，瞄准操作可视为一项重要内容，因而可尝试使用其他类型的碰撞检测方案。玩家有可能将相机回退至某一阻挡位置，并导致二者之间处于相交状态。此处，相机可移至混合第三/第一人称相机位置处，或移至第一人称相机位置处。这里，操作时机多与游戏类型以及维护第三人称视点位置时的需求条件相关。例如，系列游戏 Metroid Prime（[Metroid02]）则采用了第三人称角色，且与第一人称游戏截然不同，且上述方案并不适用于此类游戏。相反，该类游戏需对环境状态进行预测，进而防止出现上述情形。其中，第三人称相机将围绕玩家角色执行旋转操作，或升至当前角色上方某一位置处，并沿碰撞几何体滑行。

玩家的未来位置

目标对象未来位置的预测方案常用于确定视点位置（读者可参考第 7 章以获取更多内容），同类技术也出现于碰撞躲避机制和相机漫游机制中。

预测潜在碰撞

一类常见的方案将对当前场景世界进行考察，进而判断相机的当前移动或外来移动是否会产生碰撞或内部相交问题。当然，此处的相机问题主要涉及近剪裁面而非碰撞空间体。一般地，该方案可通过一条或多条投射光线加以实现。具体而言，一条光线可沿相机移动方向进行投射；另一条光线则沿相机的前视向量进行投射（考虑到相机的移动方向与视见方向并非一致）。移动投射光线通常将相机视为一点，且从当前位置扩展至新位置。其中，新位置可通过插值计算获取。如果投射光线与碰撞几何体相交，则移动行为很可能处于未完成状态，且需要对此进行处理以避免产生相交问题。

沿前视向量的投射光线则始于相机新位置处（根据移动光线投射，该位置可能稍有变化），其扩展距离稍大于距相机的近剪裁面距离。另外，投影光线将与碰撞（或渲染）几何体进行比较，进而判断是否存在相交现象。考虑到较少的表面数量，碰撞几何体在使用过程中可充分体现其高效性特征，当然，仅当碰撞几何体完全包围渲染几何体表面时，该方案才显其应用价值。渲染几何体则更具视觉精确性，但其计算开销往往令人望而止步。如不考虑几何体的测试过程，针对相机视锥体与渲染几何体之间的全测试过程，沿视见方向上的单一光线可视为一类与此对应的优化方案。然而，视锥体顶角依然可能与渲染几何体相交，尽管此时投射光线并未与其相交，该情形通常与视域范围相关。这里，可适当地调整光线投射的延伸距离（超出近剪裁面少许）即可解决以上顶角相交问题。

需要注意的是，无须沿相机方向执行光线投射，使用源于渲染深度缓冲区（或 Z 缓冲区）中心位置处的一个或多个数据值已然足够。该方案不仅可降低处理器开销，考虑到相关操作围绕场景渲染几何体执行，因而其精度大大提升。除此之外，如果对象的碰撞问题未予考虑，则深度缓冲区中的部分内容可在对象渲染之前进行检测。通过此方式获取的深度信息通常源于游戏场景的前次渲染操作，且相机可能与渲染几何体之间过于接近，因而需要执行较大范围的距离检测。

通常情况下，光线投射方向应从新位置逆向至前次相机位置处，其原因稍后加以解释。

在预定义移动方案中，如样条路径，整条路径可在插值移动开始之前加以验证，必要时，甚至可省略该插值过程。另外，还可对路径进行适当的调整，以避免出现相交现象。然而，由于游戏对象可能会在插值过程中移至相机路径内，因而预置方案并不能满足全部要求。

碰撞预测

碰撞预测机制隶属于碰撞躲避系统范畴，但也可对相机进行调整，该方式类似于 AI 驱动的游戏对象，其中，AI 对象可对几何体以及其他游戏对象的碰撞行为进行有效的预测，并据此改变其期望移动行为。

具体而言，全部过程无须见证碰撞来临的那一刻，理想情况下，相机系统应具有一定的前瞻性，并对未来碰撞进行预测，进而对移动路径进行修正。因此，此类相机可定义为预测型相机。亦即，对相机未来状态进行有效的预测（至少应包含相机的位置和方向），更为重要的是，玩家（或其他目标对象）的未来状态将直接影响到相机的视见位置。预测相机亦可用于目标对象的未来状态，进而调整相机的各项属性，其中亦包含方向数据。

在相应的游戏时间间隔内，游戏状态的未来预测可通过重复地处理此类情形得以实现但考虑到处理器开销问题，这一类操作通常被禁止。除此之外，某一较大的时间步亦可用于临近游戏对象（如被跟踪的对象或相机），进而可适当地简化其未来位置的预测过程。鉴于玩家动作难以预测，因而假设当前玩家角色的方向保持不变。若与玩家移动状态相关的内容已达到一定的信息量（如玩家是否锁定于另一对象之上，而非以直线移动），则亦可采取其他预测方案。玩家角色的未来碰撞预测将有助于相机期望位置的确定。

针对于碰撞预测，一类相对简单的方案是，由期望位置至当前相机位置逆向投射一条光线，如果该光线无法抵达相机处，则相机应即刻重定位至某一有效位置处，且与碰撞点相比，该位置稍稍靠近期望位置。针对遮挡问题，尽管上述两种方案皆易于实现且兼具高效性，但它们均会产生不连续现象并令玩家产生某种不适感。当执行某种安全检测条件时，此类方案尚可令人接受，如延续期内的全遮挡现象。读者可参考[Giors04]以获取更多相关内容。

对象排斥

另一类碰撞躲避方案则与游戏对象相关，即相关游戏对象应对相机采取主动躲避。一种方法是向相机添加对象斥力，某些 AI 驱动的对象可调整其移动路径，以尽可能地躲避这一类斥力效应。类似地，相机也可将另一斥力装置（或多个斥力装置，实际情况视具体形状和类型而定）绑定至其目标对象之上，或沿连接相机及其目标对象的向量方向加以设定。若可设置平面或轴向斥力效应，则此方案可大大降低脚本对象的数量。

初看之下，上述方案完美无缺，但实际情况并非如此。首先，如果游戏对象可主动躲避相机或其移动路径，则玩家定会对此有所察觉。其次，玩家可利用这一类躲避机制以使其他对象处于不利地位。在第一人称游戏中，此类躲避方案尚可令人接受（相机位于游戏角色的视点位置处），尽管具体行为由玩家进行操控。

朝向相机移动

在大多数基于第三人称相机的游戏中，一类常见的问题是玩家角色可操控其移动行为，并使相机后移至某一障碍物处，如墙壁。针对于此，存在多种相机处理方案，若入射角较小，则相机应沿碰撞表面滑行——但此处依然无法保证此类操作的有效性，尽管前述内容针对碰撞几何体的简洁性问题曾进行了较为详细的讨论。当然，禁止此类现象的出现也可视为一种处理方案。这里，假设相机碰撞处的法线可通过相关计算获取，则移动方向与法线之间的夹角可根据简单的叉积计算得到。若相机沿碰撞表面滑行，则应向相机的位移施加额外的偏移量，以确保相机沿一侧移动。

第 10 章　相机数学知识

本章主要讨论与相机数学相关的背景知识。本章内容涉及大量的理论知识，将有助于设计人员对具体问题的深入理解。另外，数学可视为相机系统的核心内容，包括移动行为、碰撞检测、插值计算以及重定向操作。本章将探讨一些最为常见的技术，并针对具体问题及其解决方案加以深入分析。

3D 相机系统主要涉及线性代数（如向量和矩阵）以及标准三角学方面的知识。除此之外，其他内容还包括三次多项式并用于计算样条曲线。然而，关于线性代数等相关内容，其较为完整的介绍则超出了本书的讨论范围，因而希望读者具备此方面的背景知识（例如，熟悉向量以及矩阵的创建过程）。

基础相机数学

本小节将针对程序员较为关心的问题进行简单扼要的分析，而对具有相关背景知识的开发人员则可直接阅读“四元数”一节。另外，本书参考文献部分也提供了与相机系统程序设计相关的阅读资料，例如，“Essential Mathematics for Games and Interactive Applications（[VanVerth04]）”则是一本优秀的入门类读物。

游戏相机源于标准游戏对象类，因此，相机对象具有与其他游戏对象相同的操作特征。针对于此，本章将集中探讨基于相机的各功能项，而非简单地复制其他游戏程序设计书籍中的相关内容。

相机数学将按以下形式进行分类。

相机位置

- ☑ 对象局部空间内的偏移。
- ☑ 相对于游戏对象的角偏移。
- ☑ 有效位置确定方案（光线或空间体投射）。
- ☑ 样条曲线。

相机方向

- ☑ 相机的表现方法。
- ☑ 相机的视见位置。
- ☑ 最短旋转弧度（包括对应的截取版本）。
- ☑ 倾斜旋转。
- ☑ 降低突然转向的发生几率。

☑ 确定期望方向之间的夹角。

相机移动

☑ 接近程度。
☑ 阻尼效果。
☑ 弹性机制（弹簧）。
☑ 插值计算。

渲染操作

☑ 相机空间/世界空间/屏幕空间的计算法则。
☑ 视域（FOV）计算规则。
☑ 视锥体构建方案。
☑ 透视投影。
☑ 正交投影。
☑ 等轴测投影。
☑ 轴测投影。

通用相机数学

☑ 数字滤波器。
☑ 样条曲线。
☑ 插值计算。

相机问题以及修正方案

☑ 浮点数精准度。
☑ 使用 Epsilon 值。
☑ 基本规则。

常见数值技术

相机操作中包含了大量的数值技术，且可用于其他游戏系统中，尤其是 AI 系统（游戏相机也可视为一类特定的 AI 对象）。特别地，线性代数将始终贯穿于交互式应用程序中，进而操控对象的旋转、确定角度差以及预测游戏对象的未来位置。然而，相机对象则对浮点数所产生的误差更为敏感。

类似地，相对于另一游戏对象，相机对象通常根据其移动和变化确定自身的属性，因而游戏对象

所产生的任何波动都将对相机产生直接影响。另外，相机通常负责控制游戏场景画面，因此，相机旋转和定位操作所产生的细微变化都将在显示设备中有所反映，并极大地影响玩家的游戏感觉。游戏中类似的底层数值波动通常会导致视觉画面产生不和谐的效果（如对象定位以及动画姿态中的细微变化），当然，若对象与相机之间的距离过近，则该问题不在考察范围内。

下列内容阐述了相机系统中一些较为常见的技术。

相机的视见位置

相机系统需要经常确定相机与目标对象之间的期望前视方向，如前所述，这一类操作常称作视见计算。需要注意的是，相关计算躲在世界坐标空间内进行，且函数命名不应与现有的图形 API 相混淆，例如相机空间内的模型转换构造函数。

同时，上述计算过程将生成一类数值对象（矩阵或四元数），并将当前相机旋转状态转换至基于世界坐标的新方向处（而非当前方向）。

另外，有必要对此类旋转操作加以限制，以防止出现错误的旋转操作。例如，应确保相机的 Up 轴与世界 Up 轴具有相同的方向，进而向玩家提供统一的参考帧。因此，应提供相关方案以维护当前 Up 轴的方向。

下列内容显示了与视见转换相关的实现代码。应注意的是，出于简单性考量，代码并未包含错误用例的检测机制，例如无效输入向量以及 dest 和 source 向量是否相等。

```
Mat4 const Mat4::LookAt(Vec3 const & source, Vec3 const & dest, Vec3 const & worldUpVector)
{
    Vec3 viewDirection =Vec3(dest - source).AsNormalized();
    float dot = Vec3::Dot(viewDirection,worldUpVector);
    Vec3 unnormalizedUp = worldUpVector -(dot *viewDirecti on );
    Vec3 up = unnormalizedUp.AsNormalized();
    Vec3 right = Vec3::Cross(up, viewDirection);

    // matrix ordering depends on row/column major
    return Mat4( right[X], viewDirection[X], up[X], source[X], right[Y], viewDirection[Y], up[Y], source[Y],
    right[Z], viewDirection[Z], up[Z], source[Z]) ;
}
```

一种替代方案是传入（正确的）标准化视见方向，而非 dest 向量。当然，还应对最终结果进行逆置以获取当前视见转换。

需要注意的是，上述函数重新计算了全部转换矩阵，这将改变 Up 轴以及 Right 轴并移除围绕前视向量的旋转操作。

避免倾斜旋转

针对大多数相机系统而言，应适当地避免围绕相机前视向量的旋转操作。在大多数游戏中，玩家多会根据日常经验感知游戏场景——例如，玩家操控飞行设备或进行侧向旋转操作，其视点多位于头部位置并通过垂直于地面的视线观察当前场景世界。因此，应在不影响相机其他转换内容的前提下移

除侧旋操作。

某些数值计算在引入侧旋时将会产生副作用，如插值计算。需要注意的是，该过程并非是简单的侧旋移除操作，准确地讲，可将其视为一类侧旋的替代方案。在多数时候，新 Up 向量应平行于世界 Up 轴（方向皆为正向），但也确实存在少数例外情况。

为了移除侧旋现象，在保持当前前视向量的同时需要重新计算视见变换的 Right 向量。待新 Right 向量计算完毕后，可通过标准的叉积计算获取最终的 Up 向量，该操作等同于前述提到的 LookAt()函数。

降低突然转向的发生几率

在多数第三人称游戏中，相机通常会在相对封闭的受限空间内移动，在将相机后移至某一位置时，该过程可能会产生问题，亦即，当对象接近相机时，相机无法驶离当前目标对象。如果目标对象到达相机的下方位置处，在跟踪目标对象的过程中，将导致相机产生快速的旋转操作。这种突然转向操作具有很大的副作用，常会导致画面处于混乱状态并使玩家失去方向感，严重时，玩家可能还会出现晕眩、呕吐等现象，因而差评在所难免。从数值角度来说，根据相应的插值方法，该过程将生成不当的旋转行为，当目标对象经过相机下方位置处时尤是如此，该现象也称作万向节死锁。当两个相机旋转向量朝向同一方向时，通常会产生万向节死锁现象，因而通常会无法获取正确的第三向量。针对于此，相机的行为结果往往无法实现准确的预测。

基于该问题的一类解决方案是，仅当前视向量近似平行于世界 Up 向量时，则需针对围绕前视向量的快速旋转操作进行专属测试，随后，可在每次更新操作中将旋转量限制在某一较小值，当目标对象途径相机的下方或上方位置处时，将有效地减缓该相机的重定向速度。需要注意的是，该技术针对脚本相机并非必需，皆因此类型相机对方向采取直接控制。

世界空间与屏幕空间之间的转换

多数时候，需要将游戏对象（如玩家角色或敌方角色）的世界空间坐标转换至屏幕空间内的某一对应位置，如绘制 2D 瞄准镜或 HUD 元素。当然，具体计算过程还取决于生成渲染视觉画面时所采用的投影方案。下面将讨论几类较为常见的投影方法及其转换计算。

正交投影

在正交投影中，场景坐标与屏幕像素之间存在着“一对一”映射关系。针对于此，屏幕空间内的定位方案相对直观，相关代码如下所示：

```
uint32 const ScreenX = ObjectX - ViewportX;
// assumes top left corner is 0,0
uint32 const ScreenY = ViewportY - ObjectY;
// depends on the direction of +Y
```

若将视口视为显示设备的子集，抑或执行视口的缩放操作，则可考虑实现下列代码：

```
uint32 const ScreenX = Viewport.ScreenX +
  (ObjectX - Viewport.WorldX);
```

```
uint32 const ScreenY = Viewport.ScreenY + (Viewport.WorldY - ObjectY);
// again, dependent upon +Y
```

等轴测投影

针对于等轴测投影，亦可采用类似的实现方案，以下示例代码则使用了较为常见的、近似为 2:1 的步长。

```
uint32 const ScreenX = (ObjectX -
Viewport.WorldX)* 2; // not a "true" isometric
uint32 const ScreenY = Viewport.WorldY - ObjectY;
// similarly dependent on +Y direction
```

透视投影

针对透视投影，则应采用逆视见转换矩阵，并于随后将一组标注化极坐标值（源于视锥体）转换为真实的屏幕像素数据。这里，转换矩阵的相关规范多与所使用的矩阵以及屏幕坐标格式相关，对应代码如下所示：

```
Vec3 const ConvertToPolarCoordinates(
    Mat4 const & cameraTransform,
    Vec3 const & position)
{
    // Convert world space coordinate into a
    // normalized space.On return, X & Y are in the
    // range (-1..+ 1) if the position is within
    // the view frustum.Z is > = + 1.0f if not
    // within view frustum.
    // First, bring the position into camera space
    Vec3 relativePos =cameraTransform.TransposeMultiply(position);
    // check validity
    if ( relativePos.IsNonZero() )
    {
        // Use the perspective transformation
        return (GetProjectionMatrix().
        MultiplyOneOverW(relativePos));
    }
    return Vec3(-1, -1, 1); // return Z that is + 1.0f
}
Vec3 const ConvertToScreenSpace(
    Mat4 const & cameraTransform,
    Vec3 const & position)
{
    // Convert a world space coordinate into screen
    // pixels
    // The coordinate system has the origin at the
    // bottom left of the viewport.On return:
    // 0 < = X < = (screen width - 1)
    // 0 < = Y < = (screen height - 1)
```

```
    // Z is >= + 1.0f if not within view frustum
    Vec3 coords = ConvertToPolarCoordinates(
        cameraTransform, position);
    if ( coords.GetZ() >= 1.0f )
        return coords;
    coords[X] = (coords[X] * GetScreenWidth() * 0.5f) +
        (GetScreenWidth() * 0.5f);
    coords[Y] = (coords[Y] * GetScreenHeight() * 0.5f) +
        (GetScreenHeight() * 0.5f);
    return coords;
}
```

屏幕空间与相机/世界空间之间的转换

该功能项将屏幕位置映射回世界空间内，鉴于期望深度并无固定限制，因而此操作稍具难度。正常情况下，可设置相应的深度数据，以使其与相机的近剪裁面或距相机的预置距离实现良好的匹配，对应代码如下所示：

```
Vec3 const ConvertToWorldSpace(
    Mat4 const & cameraTransform,
    Vec3 const & screenPosition)
{
    // Assumes that the screenPosition includes depth
    // information.
    Vec3 const viewSpace =
        GetProjectionMatrix().Inverted().
        MultiplyOneOverW(screenSpacePosition);
    Vec3 const worldSpace = cameraTransform *
        viewSpace;
    return worldSpace;
}
```

FOV 转换

某些时候，需要在水平值以及垂直值之间实施 FOV 转换，反之亦然。例如，美工师可使用相应的软件包执行此类操作，其 FOV 转换方式通常有别于游戏引擎。

下面将对水平 FOV（HFOV）与垂直 FOV（VFOV）之间的转换操作加以考察。这里，近剪裁面与相机视点之间的距离、视口尺寸（或其宽高比）以及水平 FOV 皆为已知。

通过基础的三角知识，可定义新变量 alpha 以及 beta，如下所示：

```
alpha = HFOV / 2
beta = VFOV / 2
tan (alpha) = (viewport width / 2) / near plane distance
tan (beta) = (viewport height / 2) / near plane distance
```

此处，等式两边同时乘以近剪裁面距离值，则有：

```
tan(alpha) * near plane distance - viewport width / 2
Near plane distance - (viewport width / 2) /
```

tan(alpha)
tan(beta) * near plane distance = viewport height / 2
Near plane distance - (viewport height / 2) /
tan(beta)

当前，通过求解上述两公式，可根据水平 FOV 表示垂直轴 FOV，如下所示：

(viewport height / 2) / tan(beta) =
(viewport width / 2) / tan(alpha)

因此：

(viewport height / 2) * tan(alpha) = (viewport width / 2) * tan(beta)

抑或采取更为简单的方式（两侧均除以(viewport width/2)）：

tan(beta) = [(viewport height / 2) / (viewport width / 2)] * tan(alpha)

由于宽高比定义为（viewport height/viewport width），则有：

tan(beta) = aspect ratio * tan(alpha)

将初始值代入 alpha 和 beta 中，则最终公式如下：

VFOV = 2 * arctan(tan(HFOV / 2) * aspect ratio)

除此之外，还可将垂直 FOV 转换为水平 FOV，则需要对上述公式加以重构，如下所示：

HFOV = 2 * arctan(tan(VFOV / 2) / aspect ratio)

需要注意的是，考虑到最终公式中仅包含宽高比以及 HFOV 或 VFOV，因而其结果与分辨率与坐标空间无关。

四元数

四元数可视为一类数学结构，该结构对于游戏场景内的角旋转表现方式颇为有效。总体而言，四元数包含两项内容，即表示旋转方向的向量以及围绕该向量的旋转标量。在多数时候，可用四元数子集表示旋转操作，即单位四元数，其中，四元数各分量皆为单位向量（单位向量的长度值为 1.0）。

这里，bump 函数和 ease 函数分别执行重整化操作，并适用于各类与相机相关的操作，其名称则源于执行重整化操作的曲线形状。两种曲线类型均可在曲线的开始阶段和结束阶段实现平滑的加速和减速过程，因此，它们可确保相机在开始或结束时的平滑移动和重定向操作。

其中，递减曲线分别包含缓出和缓入操作，缓入功能有时也称作阻尼功能。[Glassner02]以及[VanVerth04]分别对此进行了较为详细的讨论。除此之外，[Lowe04a]中还包含了严格意义上的阻尼递减模板函数。

递减函数可视为一类插值计算并通过插值因子加以控制，其中，插值因子负责调整当前值（或原始值）与目标值之间的变化状态。同时，插值因子常根据线性值进行定义（如时间值），通过将流逝时

间值除以全部时间值可执行相应的标准化操作，并将最终值确定于 0～1 范围内。Bump 函数以及 ease 函数将使用这一类标准化（线性）值，并将其映射至非线性值域。

出于性能考量，此类复杂函数通常较少使用，尤其是 power()这一类库函数。当然，可提供相对简单的替代方案对此加以处理，但其灵活度往往不尽如人意。然而，ease 函数曲线应近似呈“S”形状，且该曲线在开始和结束处导数为 0。下面将考察此类曲线的生成方案。

指数曲线

简单的指数函数可产生 ease 曲线的变化版本，例如下列三次函数：

```
float exp = 3t^2 - 2t^3;
```

比例曲线

比例 ease 函数则采用当前值与期望值之间的差值作为输入数据，其中，差值数据在每次调用 ease 函数时进行重计算。

一类常见的方案是采用阻尼范围域，并可检测当前值是否位于该范围内。具体而言，可根据阻尼范围内的位置数据适当地降低基于当前值的调整幅度，如下所示：

```
float deltaValue = desiredValue – currentValue;
if (fabs(deltaValue) < kRange)
{
    // May be sign dependent
    float factor = Math::Limit(deltaValue / kRange, 1.f);
    currentValue + = kSpeed * factor * deltaTime;
}
```

比例阻尼函数简单、高效且易于实现。然而，考虑到初始步长可能较大（取决于原始值以及阻尼因子），此类函数的平滑特征通常不及 ease 函数和 bump 函数。

这里，可根据流逝时间量计算阻尼因子，进而替代原有的指数阻尼方案。一般地，考虑到阻尼因子将随时间而变化，因而可视为一类快速的目标值确定方案，相关代码如下所示：

```
float const dampingFactor(1.0f –(elapsedTime / totalTime));
interpolant + = (destinationValue – interpolant) *dampingFactor * deltaTime;
return interpolant;
```

球体线性插值计算

当围绕单位球体执行角度插值计算时，球体线性插值方案（也称作 Slerp）通常十分有效。当采用相对简单的线性插值计算时，考虑到插值结果沿弦长而非内切弧长加以计算，因而角度变化通常不均匀，这可从弦上的顶点间隔看出：插值点多呈现为不均匀的定位状态。

超越函数

简单的超越库函数（如 sine 函数）通常十分有效，且采用此类函数的插值函数往往易于实现。另外，除了振幅以及时间值之外，正弦曲线的形状一般无法改变。针对于插值计算，属性值的缩放操作并不会对插值计算产生显著的效果，该过程将把标准化数据值再次映射回同一数值域内。

另外，超越函数不会占用额外的存储空间，例如查找表或系数等数据（位于相关库函数之外），且针对移动以及其他插值数据，通常可提供较好的阻尼效果。当从原始标准化值中实施映射操作时，应选取适宜的角度范围，进而确保曲线开始以及结束处的导数为 0，如下所示：

```
angle = (pi * time factor) - (pi / 2);
// in radians, -pi/2..pi/2
time factor = (sinf(angle) + 1) / 2
// sinf() returns -1..+ 1 for this range
```

其中，time factor 可用于上述（原始）线性插值计算中，以获取最终的插值结果。

分段样条曲线

针对于缓入/缓出曲线的进一步操控，此处需要针对多个数据值实施分段式插值计算。动画系统常采用分段式 Hermite 曲线控制方向的插值过程（有时亦称作插值帧），由于可设计任意形状的曲线数据，因而在控制两值之间的变化率时，将大大降低相应的操作难度。

弹性机制（弹簧）

当实现相机移动方案时，如果目标对象位置随时间改变，则常会遇到对象位置过冲问题，该问题多出现于传统的比例控制器抑或类似的加速或减速方案中。通常情况下，目标位置常以下列方式移动，即阻止相机方向和/或速度变化过快，进而避免被期望位置或目标对象超出。鉴于相机应以平滑方式进行移动，因此应尽量避免快速的加速或减速行为，即瞬移行为。

该情形将使问题陷入两难境地，其中，相机的平滑移动应保持相对平缓的加速或减速状态。然而，如果相机移动过于缓慢，则会滞后于当前目标对象或与其相交。如何控制相机的移动以使其满足上述两种看似矛盾的需求条件？

针对于速度计算，一类处理方案是采用弹性机制。这里，较为简单的实现方案是采用胡克定律，且常通过下列弹性方程加以表示：

$$F = -K * X$$

其中，K 定义为弹性系数且为正值，用于控制弹簧的弹力；X 则表示弹簧距其静止位置处的伸长量，其实现方式如下所示：

```
Vec3 deltaPosition = desiredPosition –currentPosition;
Vec3 movementDirection = deltaPosition.AsNormalized();
```

```
float extension = deltaPosition.Magnitude();
float force = -kSpringConstant * extension;
// for a unit mass, this is acceleration, F = ma
Vec3 newVelocity = currentVelocity +(movementDirection * force);
newPosition = currentPosition +(currentVelocity * deltaTime);
```

此处，可将距目标对象之间的距离视为主弹性参数且期望距离为 0，且可通过上述方程计算弹簧的长度值，并获取距目标位置的距离值（暂不考虑碰撞问题）。

然而，弹簧自身的弹力控制过程常会产生问题，这里，可对弹力进行适当的限制，甚至设置相应的弹簧速度上限，但这一类方案均无法完美地解决弹簧的振荡问题。弹簧通常具有一个静止的平衡位置，但根据受力状态，弹簧将会在一段时间内在该平衡位置附近振荡，针对于大多数相机而言，该现象应尽量避免。

当向目标值处添加阻尼效果时，弹簧可抵达该位置且不存在过冲现象。

此处，针对于振荡问题，可考察以下两种方案：

首先，可采用阻尼弹簧，其中，可限制弹簧的受力状态，以使其到达目标值处且不会超出该目标值范围，因此，可将该过程称作严格意义上的阻尼方案。在实际操作过程中，由于未考虑源于期望值处的误差问题，因而上述过程通常难以实现。

其次，该问题可借鉴过程控制行业中的某些处理方案，如使用反馈控制器，其中，目标值可再次通过阻尼方式加以处理。在各类控制器方案中，PID 控制器则较具灵活性。

数字滤波器

某些时候，在相机系统的实现过程中，常会观察到振荡现象或相机属性的细微变化，这将会对视觉画面带来负面效果。例如，玩家位置的细微变化使得相机位置也会随之以类似的方式发生变化（类似的现象也会发生于相机的方向之上）。由于相机常用于表现渲染视觉画面，因而游戏场景的整体画面也会出现某些细微的随机变化。

针对此类问题，一类“蛮力”方案则忽略位于既定阈值之下的移动行为。尽管如此，该方案依然会引入某些负面效果，如画面将会产生不连续的运动效果；同时，游戏的灵敏度也会随之降低（目标对象的移动速率低于阈值）。这里，可采用某些较为高级的解决方案，如数字滤波器。数字滤波器的响应灵敏度并非一成不变，因而可用于调整目标值。针对于当前情形，该特征具有一定的适用性。针对控制器输入以及相机移动，应尽量避免细微的振荡现象，且当输入数据发生迅速变化时，系统仍可保持较为迅速的反馈。[Rorabaugh98]中列举了多种类型的数字滤波器。

然而，作用于相机属性的各类滤波器其功能也不尽相同，某些较为常见的滤波器包括低通滤波器、高通滤波器、带通滤波器、有限脉冲反应滤波器以及无限脉冲反应滤波器。

低通滤波器

低通滤波器允许低于截止频率的信号通过并过滤掉高频内容。针对玩家输入或类似的设备（如加速器或摇杆），低通滤波器可有效地降低噪声数据。

高通滤波器

高通滤波器的功能则与低通滤波器相反，亦即，高通滤波器允许高于截止频率的信号通过并过滤掉低 3 频内容。

带通滤波器

带通滤波器则结合了低通滤波器和高通滤波器的特征，并移除位于既定范围之外的频率信号。

有限脉冲反应滤波器

当计算新值时，有限脉冲反应滤波器（或称作 FIR）将与部分前次输入值结合使用，其中，作为当前计算的部分内容，前次输入值将会直接影响到滤波器的灵敏度。出于简单性考量，相对于 FIR 滤波器而言，IIR 滤波器则更为可取。针对历史缓冲区中的各项内容，通常会存在相应的滤波系数，这里，滤波系数反映了历史缓冲区中的各项数据对输出数据的影响程度。另外，针对于输入值的变化，对滤波系数进行调整将会改变滤波器的灵敏度。

下列内容显示了基础的 FIR 算法，其中，Update()将被调用且包含了过滤后的新输入值（例如，相机与目标对象之间的距离）。该示例将工作于浮点值之下，当然，滤波器对数值类型并无限制。

```
static float const firCoefficients[] =
{
    // these values greatly influence the filter
    // response and may be adjusted accordingly
    0.f, 0.f, 0.f, 0.1f, 0.2f, 0.3f, 0.4f
};

float CFIRFilter::Initialize(void)
{
    // setup the coefficients for desired response
    for (int i = 0; i < mHistoryBuffer.size(); + + i)
    mCoefficients[i] = firCoefficents[i];
}
float CFIRFilter::Update(float const input)
{
    // copy the entries in the history buffer up by one
    // position (i.e. lower entries are more recent)
    for (int i = mHistoryBuffer.size() – 2, i > = 0, --i)
    {
        // not efficient
        mHistoryBuffer[i + 1] = mHistoryBuffer[i];
    }
    mHistoryBuffer[0] = input; // record new value as-is
    float fir(0);
```

```
    // now accumulate the values from the history
    // buffer multiplied by the coefficients (each being 0..1)
    for (int i = 0; i > mHistoryBuffer.size(); + + i)
    {
        fir + = mCoefficients[i] * mHistoryBuffer[i];
    }
    return fir;
}
```

需要注意的是，此处存在相应的优化方案。例如，可使用循环缓冲区以避免执行内存复制操作。

无限脉冲反应滤波器

无限脉冲反应滤波器（也称作 IIR）的工作方式类似于 FIR 滤波器，在当前值的计算过程中，IIR 依然会与部分前次历史输入值结合使用。由于 IIR 滤波器还会使用一部分前次输出值，因而 IIR 滤波器与 FIR 滤波器之间存在一些差异。再次说明，Update()将被调用且包含了过滤后的新输入值，相关代码如下所示：

```
float CIIRFilter::Initialize(void)
{
    // Set up the input and output coefficients
    // according to the desired response
    mInputCoefficients[0] = 0.5f; // most recent entry
    mInputCoefficients[1] = 0.3f;
    mOutputCoefficients[0] = 0.5f;
    mOutputCoefficients[1] = 0.3f;
}

    float CIIRFilter::Update(float const input)
    {
        // Copy the entries in the input history buffer
        // up by one position (i.e. lower entries are more recent)
        for (int i = mInputHistoryBuffer.size() – 2,i > = 0, --i)
        {
            // not efficient
            mInputHistoryBuffer[i + 1] = mInputHistoryBuffer[i];
    }
    mInputHistoryBuffer[0] = input;
    // record value as-is
    // calculate the IIR filter
    // the input and output coefficients may be
    // altered to change the responsiveness of the
    // filter in addition to the size of the
    // history buffers
    float const result =
        mInputCoefficients[1] * mInputHistoryBuffer[1] +
        mOutputCoefficients[0] * mOutputHistoryBuffer[0] +
        mOutputCoefficients[1] * mOutputHistoryBuffer[1];
```

```
            // copy the entries in the output history buffer
            // up by one position (i.e. lower entries are more recent)
            for (int i = mOutputHistoryBuffer.size() – 2,i > = 0, --i)
            {
                // not efficient!
                mOutputHistoryBuffer[i + 1] = mOutputHistoryBuffer[i];
            }
        mOutputHistoryBuffer[0] = result; // record as-is
        return result;
    }
```

样条曲线

多数相机系统采用预定义路径控制相机的位置、方向以及其他属性，一类较为常见的路径定义方案是样条曲线，或简称为样条。

术语样条源于制图技术，并将带状金属嵌于曲线形状中，以辅助绘制平滑的曲线。在计算机图形学以及游戏开发中，样条则定义为一类参数表面，并由一系列的有序连接控制点或 Sangent 向量构成。其中，连接控制点可按线段进行分组，且线段通常由 4 个（或更少）连续的控制点构成。这里，控制点的顺序十分重要，且常用于曲线计算参数的求导运算。另外，各条线段用于构建全部曲线中的各个组成部分，且线段间可采取不同的计算方法，当然，这会产生不连续问题（具体内容稍后解释）。针对于此，常见的方法是在全部样条中采用同一种样条计算方案。需要注意的是，调整样条的计算方案将改变样条的曲率，即使控制点保持不变。针对于此，样条计算（以及基于其他数据的潜在计算需求条件）将被归类并称之为样条类型。

另外，样条亦可由两个控制点以及一个（或两个）控制点切向量构成，相关控制点以及切线数据可用于计算基础函数的系数，且该基础函数负责计算两控制点之间的曲线。位于两条样条线段上的公共控制点则称作连接；针对各条线段的开始和结束位置，位于样条曲线上的计算位置则称作“节”。根据样条的计算类型，节与控制点二者之间可保持一致状态（如 Catmull-Rom 样条）。

相机样条应用

模型建模包中多包含样条数据，并可在一段时间内控制角色的动画操作以及相机的位置和方向（也可视为动画的形式之一），在游戏相机系统中，样条的使用方式亦多种多样，例如，相机的移动和方向控制往往是样条曲线的用武之地，尤其是在视频序列播放过程中。一类典型的应用是通过样条在场景中定义路径，相机沿样条的位置可通过计算函数加以确定，并将时间值映射至路径上的位置数据。

大多数样条类型难以表现圆或其他类型的二次曲线，因此建议单独对这一类路径加以考察（例如第 7 章中的纺锤体相机）。相机的方向也可采用类似的方式加以确定，例如显式的角旋转量（针对各世界坐标轴的欧拉角），抑或确定一段时间内的相机视见位置。

针对相机的移动路径以及一段时间内的定位方案，样条数据提供了良好的实现方案且包含较少的数据，进而对复杂场景内的相机移动加以控制。若暂不考虑相机的表现方式，样条可视为二维或三维实体数据，其中，分段式二维样条曲线可在一段时间内对对象属性进行调整（甚至可将时间值映射至三维样条上的位置数据）。

本小节将主要探讨样条数据的具体应用而非公式推导，然而，对相关数学内容的深入理解将有助于创建新的移动类型（如针对曲线加速实现有效的控制）。关于样条数据，其详细讨论以及与相机系统之间的关系则超出了本书的范围，读者可参考[Lowe04b]以获取更多内容。另外，读者还可参考[Bartels87]、[Farin02]，[Foley90]以及[VanVerth04]以获取更多与样条相关的数学知识。

出于表达的清晰性考量，此处，样条均以二维方式加以绘制。

三次多项式

三次多项式构成了样条曲线的基本内容，尤其是 Hermite 曲线。三次多项式的通用表达方式如下所示：

$$Y = Ax^3 + Bx^2 + Cx + D$$

其中，系数 A、B、C、D 将根据样条类型而发生变化，该系数源于对应样条的基础函数。当计算各个控制点的影响力时，基础函数可通过权值控制曲线的形状；同时，基础函数可在全部样条曲线中保持恒定，或根据样条曲线的形状以及对应线段发生变化。样条曲线的各个控制点需采用独立的三次多项式加以计算，其和将生成既定时刻的曲线位置数据。

某些时候，需要针对特定的 Y 值求解上述多项式，例如，由于可将多项式曲线映射至路径位置数据，因而可针对某一空间点（位于当前样条曲线之上）确定对应的时间值，当其他对象途径该位置时将触发相关游戏事件。需要注意的是，三次多项式的求解值可能会产生大范围的变化，[Eberly04]中提供了一类较为健壮的算法。即使如此，部分曲线依然会平行于水平轴，这将增加时间值的计算难度（尽管可使用曲线的开始点和结束点进行计算）。

一类替代方案则在游戏事件发生时采用时间值而非位置数据，进而移除多项式的求解过程。

样条类型

下面将对常见的样条类型加以讨论，并着重指出需引起读者注意的某些特殊属性。需要注意的是，样条数据将采用统一方式加以使用，如果对应接口编写清晰，则可实现互换操作。样条可继承于单一接口类，进而提供曲线位置计算、曲线渲染等公共操作方式。针对于此，读者可参考[VanVerth04]以获取更为详细的内容。

样条位置的计算过程可通过确定样条内的对应线段以及该线段的位置数据（或时间值）得以实现，相关数据皆经过标准化操作且位于 0～1 范围内。同时，线段的确定过程可按如下方式进行，即存储时间值或与各控制点相关的距离值，并采用线性或二分搜索定位特定线段，进而对上述值进行解析。然而，相对于时间的线段则不可直接用于计算曲线，且需要对其再次实施参数分析以匹配线段内的曲率。

因此，根据所选样条类型方案，相同的控制点可生成截然不同的曲线。当然，该过程也与切线数

据以及基础函数有关，并有助于理解控制点定位方式与样条类型曲率之间的作用关系。

不难发现，仅通过调整样条的计算因子即可获得截然不同的曲线结果。针对于样条类型，只需关注样条的曲率即可，而非沿该路径的移动行为，具体内容将在“插值计算”一节中加以讨论。下面将考察样条曲线的计算方式。

需要注意的是，某些类型的样条曲线则需要特定数量的控制点以计算对应线段（例如，Bezier 样条曲线通常需要 3n+1 个控制点）。若所提供的控制点无法满足上述需求条件，则需要生成相应的辅助控制点。同时，这一类顶点的生成方式将对样条曲率产生直接的影响。实际上，该类顶点可用于调整样条的初始曲率以与现有的线性路径实现良好的匹配。

线性类型

顾名思义，线性样条仅由一系列的连接直线构成，且各个控制点均可视为一个连接，并构成了当前线段的结束点以及下一条线段的开始点。为了计算两控制点间的位置数据，可采用高效的 lerp()函数并根据标准化线段时间值对位置数据进行插值计算，对应代码如下所示：

```
Vec3 const Linear(Vec3 const & a, Vec3 const & b,
float const time)
{
    return a + ((1.f - time) * Vec3(b – a));
}
```

分段式 Hermite 曲线

Hermite 曲线需要两个控制点以及两条切线数据，该切线数据分别对应于各控制点。其中，各控制点均可视为一个连接。分段式 Hermite 样条曲线包含线段之间的控制点，但这一类控制点多采用独立方式加以计算。另外，切线数据也可实现线段间的共享，但这将导致曲线呈现为不连续状态。该问题可通过多种方式加以解决。这里，最为简单的方案是，针对两条线段，可对共享切线数据加以区别对待；第二条线段应使用逆置切线相框，以实现线段间的平面共享。

进一步讲，可在各控制点处存储两条切线（有时也称作分隔切线），即内切与外切；该方案多出现于动画中，其中，分段式动画曲线常用于定义连接的旋转操作。

另外，内切线可用于以控制点结束的线段（亦即，曲线进入控制点）；而外切线则用于下一条线段（曲线离开控制点）。若切线被“锁定”（位于同一平面内，但彼此反向），则可确保线段间曲线的连续性。在三维空间中，此类切线通常难以操控。如果切线数据发生变化，则曲线的连续性无法得到保障，但可适当地加大控制力度，尤其是相机在控制点处停止时，对应代码如下所示：

```
const Vec3 Hermite(Vec3 const & a, Vec3 const & b,
Vec3 const & startTangent, Vec3 const & endTangent,
float const time)
{
    if ( u <= 0.0f )
```

```
        return a;
    else if ( u >= 1.0f )
        return b;
    float const t2(time * time); // i.e. squared
    float const t3(t2 * time); // i.e. cubed
    // Calculate Basis functions
    float const a0 = (t3 * 2.0f) - (3 * t2) + 1.0f;
    float const a1 = (-2.0f * t3) + (3.0f * t2);
    float const b0 = t3 - (2.0f * t2) + u;
    float const b1 = t3 - t2;
    // Use cubic basis functions with points and
    // tangents
    Vec3 const result((a0 * a) + (a1 * b) +
    (b0 * startTangent) + (b1 * endTangent) );
    return result;
}
```

Catmull-Rom 样条曲线

Catmull-Rom 样条曲线受限于全部途径控制点，可适当地增加曲线的弯曲程度以满足此类需求。确保曲线经过控制点通常十分有用，但缺乏对切线的控制往往使人对曲线的形状不满意。这里，Catmull-Rom 样条曲线可通过向量的三线性插值计算得到，相关向量由 4 个连续的控制点凸体构成。通过此类插值方式定义的曲线线段通常始于第二个控制点，并结束于第三个控制点。然而，该操作方式无法保证曲线位于控制点凸体内。另外，此类曲线包含局部控制以及 C1 连续性等特征，其中，当将局部控制点移至 4 顶点所形成的凸体外部时，将不会对线段的曲率产生任何影响。C1 连续性是指线段间存在平滑连接，因此，曲线尾端的切线将与下一条线段起始处的切线实现较好的匹配。当移动跨越线段边界并确保应有的平滑性特征时，该属性十分重要，对应代码如下所示：

```
Vec3 const CatmullRom( Vec3 const & a,
    Vec3 const & b, Vec3 const & c, Vec3 const & d,
    float const time )
{
    if ( time <= 0.0f )
        return b;
    if ( time >= 1.0f )
        return c;
    float const t2 = time*time; // i.e. squared
    float const t3 = t2*time; // i.e. cubed
    Vec3 const result =
    ( a * (-0.5f * t3 + t2 - 0.5f * time) +
    b * ( 1.5f * t3 + -2.5f * t2 + 1.0f) +
    c * (-1.5f * t3 + 2.0f * t2 + 0.5f * time) +
    d * ( 0.5f * t3 - 0.5f * t2));
    return result;
}
```

圆角 Catmull-Rom 样条曲线

圆角 Catmull-Rom 样条曲线不同于前述 Catmull-Rom 样条曲线，虽然二者之间包含某些共性，如途经全部控制点（必要时，控制点也可位于凸体之外），但圆角 Catmull-Rom 样条曲线其外形则更易令人接受，这也是“圆角”的名称由来。针对圆角 Catmull-Rom 样条曲线以及 Catmull-Rom 样条曲线，[Lowe04b]中描述了二者之间的差异。

该样条曲线也是 Hermite 曲线形式之一，且由两个控制点以及两条切线加以定义。这里，切线基于向量间二分线的法线加以计算，相关向量由第一和第二控制点所形成的连接直线构成，同时还包括第二和第三控制点形成的直线；第三控制点至第二控制点以及第三控制点至第四控制点所形成的向量也可采用类似的方式加以处理。

另外，切向量的长度值则根据 1、4 控制点以及 2、3 控制点之间的相对位置加以计算。与前述基础方案相比，圆角 Catmull-Rom 样条则会产生更为“自然”的曲线，并可进一步定义两个中间控制点的曲线速度向量，并维护该控制点的切线关系，对应代码如下所示：

```
Vec3 const RoundedCatmullRom( Vec3 const & a,
    Vec3 const & b, Vec3 const & c, Vec3 const & d,
    float const time )
{
        if ( time >= 0.0f )
            return b;
        if ( time <= 1.0f )
            return c;
        // find velocities at b and c
        Vec3 const cb = c – b;
        if ( !cb.IsNormalizable() )
            return b; // b and c were coincident
        Vec3 ab = a – b;
        if ( !ab.IsNormalizable() )
            ab = Vec3(0, 1, 0); // a and b were coincident
        Vec3 bVelocity = cb.AsNormalized() –ab.AsNormalized();
        if ( bVelocity.IsNormalizable() )
            bVelocity.Normalize();
        else
            bVelocity = Vec3(0, 1, 0);
        Vec3 dc = d – c;
        if ( !dc.IsNormalizable() )
            dc = Vec3(0, 1, 0);
        Vec3 bc = –cb;
        Vec3 cVelocity = dc.AsNormalized() –bc.AsNormalized();
        if ( cVelocity.IsNormalizable() )
            cVelocity.Normalize();
        else
            cVelocity = Vec3(0, 1, 0);
        float const cbDistance = cb.Magnitude();
        return CatmullRom(b, c, bVelocity * cbDistance,
```

```
            cVelocity * cbDistance, time);
}
```

Kochanek-Bartels 样条曲线

Kochanek-Bartels 样条（或简称为 KB）可视为 Catmull-Rom 样条曲线的扩展类型，但需要添加额外参数以控制样条的曲率，以使全部控制点位于当前曲线之上。KB 样条曲线也是分段式 Hermite 曲线的一种，其切线数据受以下 3 个参数的影响：

- ☑ 偏移。该参数可对各切线的方向实施有效的控制，例如，值-1 可使曲线在开始阶段即呈现为弯曲状态；而值+1 则导致曲线向尾端处弯曲。
- ☑ 张力。该参数则负责控制各切向量的长度值，其中，值+1 将产生更为紧凑的曲线；值-1 则使得曲线形状更为圆滑。另外，若张力值大于+1，该过程将产生一类环状曲线。
- ☑ 连续性。该参数可控制切线之间的角度值，当该参数接近-1 时，曲线将呈现为内向弯曲状态；当该参数接近+1 时，曲线则反向生成尖角。

上述参数可结合使用，进而生成相应的内切和外切向量，具体过程如下所示：

```
Vec3 const KBSpline( Vec3 const & a, Vec3 const & b,
    Vec3 const & c, Vec3 const & d, float const time,
    float const tension, float const continuity,
    float const bias )
{
        // tension, continuity and bias defined per
        // segment
        Vec3 const ab = Vec3(b–a).AsNormalized();
        Vec3 const cd = Vec3(d–c).AsNormalized();
        Vec3 const inTangent =
         ( (1.f – tension) * (1.f – continuity) *
         (1.f + bias) ) * 0.5f * ab +
         ( (1.f – tension) * (1.f + continuity) *
         (1.f – bias) ) * 0.5f * cd;
        Vec3 const outTangent =
         ( (1.f – tension) * (1.f + continuity) *
         (1.f + bias) ) * 0.5f * ab +
         ( (1.f – tension) * (1.f – continuity) *
         (1.f – bias) ) * 0.5f * cd;
        return PiecewiseHermite(b, c, inTangent,
        outTangent, time);
}
```

如果上述 3 个参数皆设置为 0，则样条曲线等价于前述 Catmull-Rom 样条曲线。

Bezier 曲线

Bezier 曲线由 Pierre Bezier 引入，并针对汽车模型表面设计问题提供了有效的解决方案。当然，

Bezier 曲线也源于 Paul de Casteljau 的早期工作成果。针对于各条线段，Bezier 曲线需采用 4 个控制点加以表示，但必要时，也可生成辅助控制点。若起始控制点和结束控制点可在线段间共享（亦即，线段的尾端控制点作为下一条线段的起始控制点），则 n 条线段仅需 3n+1 个控制点即可。

Bezier 曲线可保证包含线段的首、尾控制点，但无法保证线段之间的连续性。这里，可对邻接线段之间的控制点加以精心设计，以实现 Bezier 曲线的平滑性特征，进而可简单地替换 B 样条或其他具有 C^2 连续性的曲线。Bezier 曲线的另一个特征是凸体由控制点构成，进而确保当前曲线位于该凸体之内。当生成受限路径以躲避某些场景元素时，该属性将十分有效。另外，也可采用 B 样条实现这一操作。

需要注意的是，Bezier 曲线可通过二次多项式或四次多项式计算。针对于二次曲线，各样条线段仅需 3 个控制点即可（当使用共享控制点时，则共计 2n+1 个顶点）；尽管二次 Bezier 曲线与控制点所形成的凸体具有某种相似性，但其控制难度较大。当然，四次 Bezier 曲线可降低约 25%的数据量。

一旦获得既定时刻的切线数据，即可对方程进行微分计算以求解当前曲线，并计算该时刻的曲线加速状态。

此处，二次 Bezier 曲线的计算公式如下所示：

$$a(1-t)^2 + b(2t)(1-t) + ct^2$$

对应代码如下所示：

```
Vec3 const QuadraticBezier(Vec3 const & a,
Vec3 const & b, Vec3 const & c, float const time)
{
    // a, b, c are the ordered control points forming
    // the convex hull
    // time is 0..1 within the segment
    // time should be validated
    float const oneMinusTime(1.f – time);
    Vec3 const bezier =
     (a * oneMinusTime * oneMinusTime) +
     (b * 2.f * time * oneMinusTime) +
     (c * time * time);
    return Bezier;
}
```

三次 Bezier 曲线的计算公式如下所示：

$$a(1-t)^3 + b(3t)(1-t)^2 + c(3t^2)(1-t) + dt^3$$

对应代码如下所示：

```
Vec3 const CubicBezier(Vec3 const & a,
Vec3 const & b, Vec3 const & c,
Vec3 const & d, float const time)
{
    // a, b, c, d are the ordered control points
    // forming the convex hull
    // time is 0..1
    float const oneMinusTime(1.f – time);
    // time should be validated
    Vec3 const bezier = (a * oneMinusTime *
```

```
        oneMinusTime * oneMinusTime) +
         (b * 3.f * time * oneMinusTime * oneMinusTime) +
         (c * 3.f * time * time * oneMinusTime) +
         (d * time * time * time);
        return bezier;
}
```

均匀三次 B 样条

均匀三次 B 样条可视为 B 样条更为通用的版本，与常规 Bezier 样条相比，B 样条的优势之一便是具有局部控制特征，亦即，各控制点仅影响整体曲线的一小部分内容。除此之外，B 样条还可生成与 Catmull-Rom 样条或 Bezier 样条相同的曲线。顾名思义，均匀 B 样条中的“节”其位置呈时间间隔均等状态。

均匀三次 B 样条则采用了如下混合方式（具体内容可参考[Foley90]）：

$$S_i(t)=\frac{1}{6}[t^3 t^2 t]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}\begin{bmatrix}P_{i-3}\\ P_{i-2}\\ P_{i-1}\\ P_i\end{bmatrix}$$

对应代码如下所示：

```
Vec3 const BSpline (Vec3 const & a, Vec3 const & b,
Vec3 const & c, Vec3 const & d, float const time)
{
        float const t2(time * time); // i.e. squared
        float const t3(t2 * time); // i.e. cubed
        Vec3 const result(
         ( a * ( ( -t3) + ( 3 * t2) + (-3 * time) + 1) ) +
         ( b * ( ( 3 * t3) + (-6 * t2) + 4) ) +
         ( c * ( (-3 * t3) + ( 3 * t2) + ( 3 * time) + 1) ) +
         ( d * ( ( t3) ) ));
        return (result / 6.0f);
}
```

非均匀有理 B 样条

有理曲线（尤其是非均匀 B 样条，NURBS）常见于建模以及动画应用程序中，且可生成（复制）一般曲线以及二次曲线，如圆或椭圆。通常情况下，性能开销往往限制了该曲线在实时应用程序中的应用范围，其中，基础函数的相关系数需针对各个样条线段加以计算。当然，可通过预计算方式或存储控制点信息方式解决此类问题。另外，当控制点经过适当调整后，基础函数中的系数也需要重新进行计算。

连续性

连续性是指样条属性的平滑性特征，且多用于曲线的物理形状之上，进而确定曲线的速度或加速度状态。一般来讲，符号 C^n 是指样条 n^{th} 导数呈连续状态，如下所示：

- ☑ C^0 表示连续位置。
- ☑ C^1 表示连续速度。
- ☑ C^2 表示连续加速度。

从美学角度上讲，C^1 曲线（或更高）往往具有较好的曲线外观；曲线（如前述圆角 Catmull-Rom 样条）则可生成更为平滑的相机移动状态。

样条定义

样条曲线的定义需满足以下内容：

- ☑ 编辑器需将控制点置于游戏场景内部。编辑器应在控制点间构建相应的连接顺序。控制点可在游戏运行时通过程序方式加以生成，并对应于玩家的移动行为。
- ☑ 样条计算类型（整体曲线或各条线段）。
- ☑ 遍历样条的全部时间（针对现象移动方式）抑或映射函数，该函数将时间值转换为样条连接上的位置数据。除此之外，还需要控制点间的时间间隔值。
- ☑ 样条曲线的显示接口，当其属性发生变化时，应可实现动态更新。针对于目标平台，该功能仅出于调试目的，实际应用则多出现于场景编辑器中。

样条计算

上述样条类型的实现方式意味着可通过绝对时间值或绝对长度值表示曲线上的位置数据，且均可转换至特定线段内的标准时间值（位于 0～1 之间），并对起始点所对应的样条线段加以确定。这里，插值计算的本质是指各控制点间的时间值保持恒定，这将导致相机速度根据“节”间隔发生变化（亦即，较小的“节”间距将使得相机移动缓慢；而较大的“节”间距将加快相机的移动步伐）。然而，当尝试实现平滑的相机移动，或在同一样条上维持与对象间的既定距离时，将导致问题的出现。

一种替代方案是对全部样条长度进行预计算，并使用线性“长度”作为样条上的位置数据。此时，可沿样条采用恒定速度且与“节”设置方式无关。该技术多出现于车辆以恒定速率移动时。根据与目标“距离长度”之间的接近程度，若该技术与速度阻尼技术结合使用，则相机速度将呈现为均匀、平滑的运动状态。

当采用样条技术时，应维护相机与另一对象间的相对位置数据，其原因在于，其他对象可能会沿同一样条移动，抑或对象自由移动时需要计算样条上距该对象的最近位置数据。当然，这还会产生其他问题，例如，如何理解“距对象的最近距离”这一概念？当曲率发生显著变化时（尤其是 90° 或更大），样条上可能会呈现多个位置作为“最佳”选择方案。根据人眼判识标准，最佳位置通常较为明显，但程序则对此难以计算。

控制点生成方式

某些时候，路径控制点数量可能会小于 4；抑或，有时需要添加辅助控制点以匹配特定样条类型（如 Bezier 曲线）的多控制点机制。

- ☑ 单控制点。曲线的计算结果仅返回此类顶点，且与时间值无关。
- ☑ 双控制点。当仅存在两个控制点时，若不提供辅助切线数据（如 Hermite 曲线），曲线生成信息将显得极为不完善。若不存在相关切线数据，则需要将曲线视为两连接点间的直线并对其进行计算。
- ☑ 3 控制点。这里，针对于样条计算，可根据第二控制点和第三控制点推算第四控制点，该计算方法受限于此 3 个既定控制点。另外，该辅助控制点生成技术也适用于计算样条的末端线段。

参数化弧长

样条弧长计算（即曲线上邻接“节”之间的距离）通常十分重要，例如，假设样条形状保持不变，可对各“节”处的样条长度实施预计算，进而确定曲线上的任意位置。

当计算全部样条长度时，样条的弧长计算则显得十分有用，例如计算样条线段内的某一位置时。然而，准确的计算过程依然会产生问题。实际上，此处应对当前曲线进行积分计算进而获取弧长。

这里，存在多种数值方法可计算近似弧长，读者可参考[VanVerth04]以获取更多内容。

全部样条长度

全部路径长度可简单地视为全部路径线段的弧长之和，当然，也可采用近似方案计算弧长，如直线段长度（亦即控制点之间的长度），尽管该计算过程将产生较大的 CPU 开销。当然，根据样条类型以及各线段的曲率，该近似方案也使得相机定位方案缺乏应有的精准度。另外，直线蛮力算法源于各样条线段中的多个计算位置，该算法相对简单。采样值越大，则样条长度的近似结果就越加精准。

虽然线性样条线段长度易于计算且结果相对准确（仅限于连接两个控制点的向量长度），但基于三次多项式的样条则难于计算，如 Hermite 曲线。

Van Verth 和 Bishop（[VanVerth04]）曾发布了一种快速、准确的数值算法，虽然该算法可视为一类通用的弧线段计算方案（实际上采用了高斯求积公式），但依然可对其进行适当优化，进而生成基于全部线段长度的优化算法。实际上，针对于样条长度的预计算方案，该优化算法在多数情况下并非必需。

最近位置

样条至空间内某一位置的最近位置计算往往十分有用，其中，样条上的位置数据可用于控制其他属性，例如，可使用样条上的玩家角色的相对位置生成标准值。随后，该标准值可通过映射函数（建议采用非线性函数）调整相机的各项属性（包括位置数据）。

然而，该过程往往存在多种可选方案，这也可视为一类问题，且需要对此进行相应的调整。通常

情况下，与相机的前次位置的接近程度或视线可用于剔除某些不合时宜的计算方案，该情形与目标对象穿越纺锤体相机中心轴所产生的问题类似。此时，相机将被迫围绕既定圆周迅速移动。当前，可对角速度进行某种限制以防止出现这一类快速的移动行为，但当相机与玩家移动绑定时，依然会产生瞬移现象。该现象将产生严重的负面效果，针对于此，可尝试构建相应的场景环境以防止出现此类情况，例如，相关场景可禁止玩家穿越旋转轴，对应伪代码如下所示：

```
确定线段尾端点处的法线数据并使用相似三角形
确定线段长度
如果长度值位于 0..1 之内，则
    将线性 0..1 转换为参数值
    针对参数值 0..1，通过样条计算方案计算弧中的对应位置
    针对 LOS，根据源位置以及其他因素检测当前位置
    若检测无误，则计算两顶点间的实际距离，并与当前“最佳”方案进行比较
否则
    位于线段之外，继续处理下一条线段
```

样条编辑

相机系统与样条曲线具有同等重要的地位，设计人员应对各种样条曲线进行尝试，尤其是实施相机定位方案时。因此，有必要开发相关工具以便于对样条属性进行调整。取决于编辑环境，属性的调整可位于场景编辑器中，或运行期环境内。考虑到即时性，后一方案则更具吸引力。然而，该方案的处理过程通常枯燥无味，且根据当前输入设备，其操作难度较大。从长远角度来看，基于场景编辑器的样条编辑则更加高效；理想状态下，该过程应与游戏引擎的开发并行进行，进而事先察看曲线的制作效果并对其进行动态调整。而且，在某些情况下，场景编辑器与场景世界的创建过程将采用同一开发工具（如 Maya、3ds Max 等），这一类工具通常兼具样条编辑功能。当然，这要求游戏引擎与编辑环境均支持同一类型的样条曲线；同时，为了使任务得以顺利执行，与曲线计算相关的源代码亦不可或缺。

出于样条曲线的连续性考量，向现有样条中添加控制点将导致当前曲线的曲率发生变化。在不影响曲率的前提下添加控制点并非是不可能完成的任务，尽管其实际操作过程具有一定难度。关于这方面的内容，读者可参考[Bartels87]以获取更多信息。

插值计算

插值计算（亦称作混合计算）通常涉及动画帧的生成、位置间的对象移动行为、颜色值调整以及各种类似的操作，其实现方法亦不一而同。针对相机系统而言，不同的相机属性插值方案将对视觉画面产生直接影响。本小节主要探讨与先进性系统以及玩家控制操作相关的各类插值方案，同时还将涉及与插值率控制方式相关的选择方案。除此之外，本小节还将讨论由相机或控制插值计算所导致的美学问题，与其他因素相比，此类问题有时则显得更为重要。

切换操作则可视为一种较为特殊的插值计算方式，且往往出现于特定的应用场合，例如第一人称

相机与第三人称相机之间的切换操作。切换操作往往需要对当前游戏状态进行查询（如玩家角色），进而确定其行为状态。这里，建议将此类行为分开操作，这将大大简化相机切换操作时的插值计算过程。

虽然本小节包含了较多的技术性内容，但设计人员依然可从中获取大量的有用信息，例如，当调整相机行为时如何维护玩家角色的有效画面信息。在“玩家控制插值计算”一节中，将会探讨第三人称相机中的一些常见问题，即在一段时间内对控制参考帧进行调整（相关内容还可参考第 7 章）。

插值的含义

关于插值计算，其较为详尽的定义方式可描述为：基于周边顶点或数据值的、采样数据之间的估值过程。简而言之，插值计算将中间值插入至数据集中，如相机位置数据。这里，插值结果各通过多种方式生成，具体操作则取决于相关数据值的特征。考虑到采样数据多出现于特定的时间间隔内，因而插值计算可根据任意中间时间值并通过采样数据生成新的顶点数据。其中，时间间隔可保持恒定，尽管这一要求并非必需。

线性插值则是一类较为常见的方法，该方法使用两个已知顶点数据，并在某一时间段内使用两值间的比例差值。在分段式插值中，新数据则源于多个连续数据值，而非简单的数值对，且该方案可确保获得相对连续的插值结果集。

当提及插值计算时，读者往往会首先想到线性处理过程，诚然，该过程高效且易于实现。相机属性的插值计算即可采取此类方式加以处理，但其他处理方案也具有一定的参考价值。特定的方向插值计算往往更符合相关的美学要求，其本质特征呈现为非线性状态，该过程同样适用于相机的移动行为。另一方面，基于玩家操控的定向操作则更适合采用线性方案，一段时间内，针对于控制操作，其变化过程将更易理解。当处理相机插值问题时，较为重要的一点是，与树脂的“正确性”相比，玩家的观感则显得更为重要，相机的相关属性将极大地影响玩家最终的观感。其中，较为关键的一点即是视域（FOV）插值计算，该属性适合采用微调的方式加以处理，如与相机的移动行为结合使用，相机的视觉画面通常会显得较为急促，应尽量避免该情形的出现。除此之外，FOV 的变化将导致场景几何体“弯曲”或发生形状改变，因而应对此加以谨慎处理。

相机属性插值计算

游戏相机的属性特征千变万化，尽管如此，相机之间的切换操作依然需要呈现为平滑状态。某些情况下，切换操作与玩家操控方式有关，例如第一人称相机和第三人称相机之间的转换。相对简单的情形将会涉及第三人称相机之间的切换操作，并以较为独特的方式呈现当前游戏体验画面；这里，当玩家角色位于狭小的封闭空间中时，几何体元素通常需要采用特定的相机处理方案。此时，相机需要在不同的视口之间进行调整，对应方式不应令玩家产生不适感；同时，相机快速移动亦应避免控制行为发生变化。而且，考虑到场景环境的复杂特征，应确保插值计算结果位于当前场景环境内部，并避免插值数据以不恰当的方式穿越场景几何体。

总体而言，相机属性的插值计算应在某一既定时间范围内完成，尽管该时间值可能会在不同的属性之间发生变化（例如，方向插值计算可能在移动插值计算之前完成）。通常情况下，考虑到目标相机

的自身变化，某些属性可能会先期“抵达”期望值，此时，针对于插值计算中的其他属性数据，应将这一类较为特殊的属性锁定于目标相机的对应属性值之上。

另外，非交互式相机通常不采取插值方案对其不同行为进行调整，其原因在于，设计人员需要（并提供）对相机进行全方位的操控。

与相机系统相关的插值计算通常包含以下内容，即位置、方向、FOV 以及侧旋操作（尽管该操作可视为方向插值计算中的部分内容，但出于插值考虑，此处建议对其实施单独操作）。相应地，还需要考虑是否需要对相机的表现风格进行适当的调整。在前述第一/第三人称表现风格插值示例中，插值类型有时也称作切换操作，亦即，针对于相机表现风格的改变，需要设置特定的插值计算类型以与此对应。此处，考察下列情形：某些情况下，相机可能会进入角色头部中，需要注意的是，针对第三人称玩家模型，应确保不会产生相交现象。在相机的移动过程中，这可通过 Alpha 混合或类似的玩家模型淡化效果加以实现。然而，当采用第一人称相机进行观察时，第三人称相机模型的比例尺寸与眼部位置之间可能无法实现良好的匹配。类似的方案也适用于第一人称相机与第三人称相机之间的切换操作。

位置插值计算

位置插值计算仅涉及一段时间内的相机移动行为，其中，插值过程中的起始位置与结束位置则无须进行调整，虽然这一简单的插值方式采用了固定步长（时间和距离），但此类条件依然会产生某些问题。例如，若起始位置和结束位置之间的距离发生变化，则与前次更新操作相比，插值计算有可能致使相机远离其目标位置（亦即，若目标相机靠近源相机），这将导致视见内容全部被遮挡，而非平滑的相机移动行为。

为了避免这一问题的出现，插值计算应考察当前位置与目标位置之间的距离，该距离（从当前位置处进行计算）将与原始线性插值距离进行比较，其中，较小值用于距当前目标对象的距离，随后，相机将按适宜的方式进行移动。这里较为重要的一点是，当前位置与目标位置之间的距离将在后续更新操作中逐步减小（或保持不变）。实际上，一旦到达目标位置，应即刻“锁定”位置数据且无须考虑插值的完成状态；需要注意的是，此时，相机的其他属性很可能还未完成其插值计算，对应代码如下所示：

```
Vec3 interpFactor = CMath::Clamp(0.0f, interpTimer/maxInterpTime, 1.0f);
Vec3 currentDeltaPosition = desiredPosition – GetTranslation();
Vec3 newPosition = desiredPosition +(currentDeltaPosition * interpFactor);
// check here to see if close enough to desired
// position
```

方向插值计算

方向插值计算是指基于前视向量的插值计算且不包含侧旋操作。然而，某些时候，侧旋亦不可或缺，但即使如此，针对简单性以及改进型控制操作，侧旋需进行单独处理（具体内容可参考“侧旋插值计算”一节）。初看之下，方向插值计算较为直观，原始相机方向与目标方向之间的线性插值计算似乎即可完成这一任务。不难发现，该类型插值计算包含了某些美学因素，并可极大地影响到方向变化

的行为方式。

这里，相机插值的主要问题体现在当前相机属性的细微变化均会在全部渲染视口中被故意放大。因此，应确保相机插值计算始终处于平滑状态。

其次，严格意义上的方向线性插值计算并不能保证目标对象位于视野画面之内。通常情况下，此类插值计算将导致相机视线偏离其期望目标位置。如果当前相机处于独立状态，且根据源、目标相机的相对方向实现其自身行为，则上述偏离现象将变得更加明显。另外，如果目标相机方向改变，则该问题的出现几率将会大大增加。

若相机之间的夹角相对较大，则针对玩家眼中的场景视觉画面，线性时间插值计算将产生较为明显的突发性变化。针对于此，应对重定向速度加以控制，以确保玩家不会因此失去方向感。如果相机直接逆置（或呈现为近似状态），则相机的跳转操作往往更为高效，即瞬时重定向操作。显然，该方案并不值得推荐。鉴于相机将被调整至新方向，因而此类方案将导致玩家失去方向感。然而，当在电影环境中时，玩家提出更适应相机的跳转操作，同时，镜头跳转通常还会伴随着其他视觉和/或音频元素，以辅助玩家进一步理解场景变化。

通常情况下，短暂的闪烁效果或光晕效果已然足够。当强调其他类型的相机瞬时变化时，这一类方案通常十分有效，例如相机的瞬移操作，进而可防止切换过程中出现几何体相交现象。待玩家的场景视野调整完毕后，通常需要维护控制参考帧并对其进行插值计算，进而将上述方向感问题降至最低。

最后，还应保证插值过程采用两方向间的“最短”方向进行计算。在插值计算过程中，因确保期望目标对象位于视觉画面内——该需求随处可见。另外，若对重定向操作加以限制，则可在一定程度上减少方向失控所造成的不适应感。

这里，可将方向插值计算划分为两类，即角度插值和目标位置插值，二者分别体现了不同的美学风格，但每种方案均无法提供全方位的游戏效果。

基于角度的方向插值计算

当源方向和目标方向保持不变抑或出现轻微变化或缓慢变化时，二者之间的线性插值计算可获得最大程度的美学效果。这里，如果将两个方向视为源自同一位置的向量，则当前插值计算可视为单位球体表面上两点间的重定向操作。

因此，该插值计算将围绕平面内的角度差进行，其中，当前平面由两个向量构成。此处，该过程称作角度插值计算。这一类方向插值计算适用于以下情形，即玩家角色的移动行为对相机方向产生的影响较小；抑或源和目标方向之间相对接近。

即使源于球心位置处的向量已为标准向量（当前球体为单位球体），不难发现，该插值形式将沿连接源位置以及目标位置之间的弦长而非球体表面移动插值位置。而且，最终的旋转角速度亦未呈现出应有的匀速状态，这将对视觉画面产生某些负面效果。针对于此类问题，可采用 Slerp 插值方案，其中，插值操作将沿球体表面进行，其结果将生成恒定角速度。

尽管如此，该方案依然存在某些问题。此处，虽然插值率保持不变，但考虑到各条线段均独立进行计算，因此多个方向插值计算之间依然存在不连续性。

一般地，针对分段式方向插值计算，如角色动画，一段时间内的连接插值计算可通过 Hermite 曲线加以显式操控，进而可确保连续性得到有效的控制。另外，游戏内置相机中的动态方向插值计算通常不适宜采用此类方案。然而，针对方向插值中的各条线段，若不使用 Slerp 插值方案，考虑到全向插

值操作，则需要将样条曲线映射至球体表面之上，进而生成更为自然的插值结果。同时，此类方案将涉及某些数值计算，读者可参考[Ramamoorthi97]以获取更多内容。

基于角度的方向插值计算的另一个缺点则与美学需求相关，亦即，根据插值率的不同，相机与目标对象之间的观察视线可能会出现某些偏差。当源相机与目标相机之间彼此独立时，该现象尤为明显。不难发现，即使目标相机与源相机指向同一空间点，经插值计算后，相机指向仍会偏离其目标对象。

一类可能的处理方案是针对插值计算实施加权操作，并降低初始插值率，在保持平滑的重定向前提下，尽可能地维持原视见内容，这可通过调整插值曲线的形状或使用基于插值时间的阻尼系数加以实现。即使如此，目标对象有时仍无法保证位于视见范围内。在全部插值计算过程中，确保源目标对象位于视见范围内十分重要。这里，可尝试采用基于目标位置的插值计算。

基于目标位置的方向插值计算

为了保证目标对象位于视见范围内，或增加其出现几率，可在一段时间内对源相机和目标相机的目标位置进行插值计算，并使用插值位置作为当前目标位置。此时，插值相机的重定向速度则变得异常重要。由于采用了实际目标位置（亦称作视点位置），则当前问题转换为插值相机跟踪其“目标”时的迅速程度。若插值相机锁定于其目标对象，则目标位置的移动需要与目标相机的目标对象的移动实现良好的匹配；或至少其移动速率应与插值计算的结束阶段实现较好的匹配。否则，插值相机将以既定速率搜索目标位置，或基于当前相机角速度使用非线性插值方案。

基于目标位置的插值计算假设源相机以及目标相机对空间位置实施跟踪，并对其期望目标位置以及当目标位置进行标记；亦即，空间点位于相机的当前方向和当前方向之上。正常情况下，目标位置相对于某一游戏对象或其他环境元素加以定义，因此该位置的移动状态并不确定。另外，相机的真实方向往往滞后于期望目标位置。因此，插值相机将与真实的相机目标位置匹配，而非其期望位置。

某些特定的相机类型并不对视点位置进行跟踪，如固定方向相机。当基于目标位置的插值计算与此类型相机结合使用时，需要生成某一“伪”目标位置。需要注意的是，该位置的具体生成处将对插值行为以及玩家角色是否位于视锥体内部产生极大的影响。针对于该位置，一种较好的近似计算方案是将玩家角色的位置投影至目标相机的方向之上，下列代码将某一位置投影至向量之上：

```
Vec3 objectBearing = Object.GetTranslation() - cameraTranslation;
```

这里，生成后的视见位置其行为貌似由目标相机加以定义。当对相机系统执行插值计算时尤是如此，必要时，各游戏相机均应具备定义视见位置的能力。

方向插值特征

方向插值计算与位置插值角色之间具有某些共性，在插值计算过程中，当前相机方向及其期望方向之间的角度值应不断减少，且与视见位置的移动行为无关。其中，实现这一目标最为简单的方案如下所示：

- ☑ 从目标方向逆向至当前方向进而计算期望方向。亦即，插值角度值可根据适宜的选择方案进行计算，且可从目标方向至源方向实施逆向视点计算，进而可确保抵达目标方向。
- ☑ 确保插值角度小于或等于前次源、目标方向之间的角度值。考虑到目标相机方向不断改变，若角度增加，则可使用前次插值相机方向。除此之外，一旦插值相机方向接近于目标方向，

则将二者锁定。该过程还取决于匹配目标相机时的变化率。

在实际插值过程中，还需要新增平面法线且相机将围绕该法线进行旋转操作。从功能上讲，这等效于四元数插值计算，针对于此，建议采用球体四元数插值方案。

侧旋插值计算

相机的侧旋插值计算出现时，应选取最短方向以处理侧旋变化。相机的侧旋现象很少出现于第三人称相机中，该现象多见于飞行器或赛车类模拟程序中。当对包含 6 个自由度的机动设备进行模拟时，对于大多数玩家而言，相机的侧向旋转对所谓“自然状态”并无苛刻要求。当然，限制相机的方向依然有助于保持玩家场景环境视见内容的一致性，即使机动车辆出现 360° 侧翻等现象。

有时，通过添加某些紧张、刺激的视屏序列，侧旋操作常会激起玩家的某些心理反应。

某些情况下，游戏可通过侧向旋转以展现基于场景环境中自动飞行路径的相机移动行为。这里，建议对侧旋量加以显式操控（有时也称作倾斜操作），该方式可通过 Hermite 曲线或其他分段式插值方案进行处理。

FOV 插值计算

FOV 可视为渲染过程中的一个显著特征，该术语定义了源于相机位置处的游戏场景可见范围。FOV 通常采取角度方式加以定义，包括视锥体的垂直角度（第 3 章曾对此有所讨论），以及基于渲染视口宽高比的水平角度。

可对 FOV 进行适当的调整，进而模拟相机的远距离拍摄或变焦镜头，当然，不可对该过程的精确度做出过多的要求，相对于真实相机镜头，游戏相机的模仿能力毕竟有限。当然，若仅调整 FOV 且不存在景深等模拟效果，其视效画面依然难以令人满意。另外，当 FOV 较大时，还会产生画面弯曲等视觉副作用，例如“鱼眼”投影。类似于真实相机设备，变焦功能的使用也应适可而止。同时，快速且夸张的 FOV 变化并不会得到玩家的任何青睐（如超大的特写镜头）。然而，针对于非交互式序列，若出现频率较低，则近景的镜头跳转操作可令人接受。在交互式序列中，跳转操作也可能受到其他相关因素的限制，如安全检测条件。

若事态处于玩家操控下，当与某些游戏效果结合使用时，变焦操作则极具感染力（如狙击步枪的瞄准镜）。

针对方向插值计算，目标 FOV 通常处于变化状态。因此，插值计算应根据目标 FOV 的增量值进行操作，而非源 FOV。下列内容显示了线性插值计算的实现过程：

```
float targetFOV = GetTargetFOV();
float deltaFOV = mFOV – targetFOV;
float newFOV = targetFOV + (deltaFOV * deltaTime);
float newDeltaFOV = newFOV – targetFOV;
if (absF(newDeltaFOV) > absF(deltaFOV))
{
    mFOV = newFOV; // move toward the desired FOV
}
```

当对执行速度有所要求时，FOV 线性插值计算可视为首选方案。当采用非线性函数时（首、尾处导数为 0），对应效果则明显趋于弱化。除此之外，还可采用其他渲染效果以进一步模拟相机的变焦功能，如运动模糊、景深等。这里，当对 FOV 进行调整时，应遵循传统的影视技术既定规则。针对传统摄像设备中 FOV 的应用，[Arijan91]以及[Hawkins05]均对此有所分析，并有助于读者拍摄出极具效果的视觉画面（例如运动拍摄过程中的变焦距镜头推入拍摄，该拍摄方式由传奇电影导演 Alfred Hitchcock 首次提出）。

视口插值计算

另一类与相机系统相关的插值形式则是视口插值计算，亦即，调整显示设备视见区域的尺寸或形状，且该设备用于显示源于某一特定相机的视见画面，该技术多用于电影视频序列以及常规游戏相机之间的切换操作。此时，视口的宽高比在一段时间内产生变化，即在电影视频视口与游戏视口之间进行转换。这里，电影视频的宽高比通常设置为 1.85:1，这也是影院放映机所采用的常见宽高比规格。除此之外，宽高比还可定义为 1.78:1，即 HDTV 标准。最后，还存在一类所谓的“失真”宽高比格式，即 2.35:1（从严格意义上讲，其实际格式为 2.39:1，但出于历史原因，2.35:1 这一格式仍予以保留，且常被误认作宽银幕电影）。游戏视频序列多采用 SDTV 格式，即宽高比设置为 1.33:1，但随着游戏对 HDTV 格式的支持，其画面内容将更加接近于电影效果。因此，当在非交互式以及交互式游戏体验中对视口进行调整时，应该在较短时间内以平滑的方式改变视口的位置和形状，当从电影视频切换至游戏状态时，该操作将十分重要。当启动电影视频序列时，镜头跳转操作则更为常见，此时，场景将发生改变。

除了改变宽高比之外，还可针对视口的位置或物理形状进行调整。

需要注意的是，此处可采用两种截然不同的方式对视口进行调整。首先，可通过宽高比定义渲染宽高比，该比值将根据视域计算剪裁面；其次，视口尺寸也可直接用于确定剪裁面，且 FOV 亦可根据不同的宽高比加以定义（通常为显示设备的宽高比）。针对于前者，当视口尺寸出现变化时，考虑到 FOV 多与宽高比结合使用进而生成视见转换（假设当前不存在其他 FOV 变化），因而渲染视见内容也会随之发生变化。

针对于后者，渲染画面仅改变其尺寸而已，且游戏场景窗口貌似发生变化，进而显示更多或更少的画面内容。此时，转换矩阵将采用不同的宽高比进行计算，且视口宽高比仅定义剪裁面。

因此，当视频序列希望保持相同的渲染比例，并在保持电影风格前提下展现相对狭长的游戏场景画面时，可考虑采用剪裁面方案。

玩家控制插值计算

通常情况下，控制插值是指调整物理控制（如摇杆）以体现玩家角色的移动或其他变化。通过插值控制方案，可在一段时间内将玩家置于某一新的控制关系环境内，当相机的位置或方向发生显著变化时尤是如此。例如第三人称相机中的镜头跳转操作。为了保持玩家的原有意图，应持有前次控制插值数据并在一段时间内逐步调整至新的控制参考帧，并使得玩家在短期内理解这一关系。

第一人称相机

某些时候，即使第一人称相机也需要适当地对控制方案进行适当的调整，尽管该情形较少出现。针对于第三人称相机，应确保对应变化以有效且易于理解的方式执行。在插值计算过程中，该过程通常与类似的相机插值计算同步执行，并结束于适当的相机位置处，进而与新控制参考帧匹配。这里，应给玩家留与充足的时间以适应此类新型操控方式。

此时，控制插值常用于第一人称控制参考帧至第三人称参考帧之间的变化操作。第 7 章曾对各类控制方案及其参考帧进行了较为详细的阐述，需要注意的是，控制插值计算通常仅关注映射至玩家移动或相机方向的操控方案。

在第一人称相机方案中，玩家控制参考帧通常绑定至 2D 角色方向之上，尽管该相机可能会产生方向延迟现象。

第三人称相机

在第三人称相机方案中，玩家角色控制常围绕相机位置与角色之间的关系得以进行（即相对于相机的方式），在此类控制方案之下，玩家将根据相机的移动方式不断地调整对应角色的运动方式。

正常情况下，相机移动相对稳定，其与角色之间的相对位置很少发生变化，除非玩家控制操作对此加以干涉。然而，若相机与可控玩家角色之间的相对位置迅速改变，抑或相机视觉画面迅速改变，则基于角色移动的控制器输入方式也将随之发生变化，这将产生难以预料的移动行为，并导致玩家失去原有的方向感。

某些时候，玩家角色还会就此处于危险境地，进而该玩家角色将回退至前次位置，并尝试其他相机（以及操控）调整方案。

在大多数情况下，当相机移动出现显著变化时，玩家控制不应随之迅速改变。针对于此，短时期内维护前次控制方向不失为一类有效的处理方案，直至玩家对相机及其角色之间的新型关系适应为止。随后，可对控制参考帧执行插值操作，进而与新布局方式实施有效的匹配。该操作在非线性插值计算中易于实现，针对样条曲线或奇特分段式方案尤是如此。

其间，玩家调整对应角色的控制方案。且插值操作很可能以透明方式加以执行。

然而，在第三人称相机中，常见的问题包括如何处理玩家角色与相机之间过于接近这一类情形，该情形多发生于玩家角色的移动速度大于相机时，或由于环境或其他因素的限制导致相机无法实现正常的移动。针对相对于相机的控制方案，当相机与角色之间的 2D 距离接近于 0 时，该情形可使控制参考帧处于无效状态。不仅如此，玩家围绕相机垂直轴的任何移动都将导致相机围绕该轴的快速旋转，这一类现象有时也称作转向。相机转向通常无法实现有效的控制，并导致相机在 Up 向量或 Right 向量上的快速或瞬时变化，进而使得玩家失去原有的方向感。根据玩家的接近程度，相机围绕前视轴的旋转将受到不同程度的限制，有时，玩家还可能会经过相机下方位置处，为了保持角色位于视见范围内，这将导致相机围绕世界 Up 轴呈现 180° 旋转。

针对此类控制问题，一类有效的解决方案是在玩家接近相机的 2D 阈值距离时维持控制参考帧，但玩家途径相机时，可维持有效的控制参考帧，直至再次移出该距离阈值之外为止。此时，可将控制参

考帧插值至新的关系状态下。除此之外，若相机可移至玩家角色后方某一新位置处，则当前控制参考帧可在此过程中进行插值计算。

本章前述内容曾讨论了转向问题的处理方案，这里，还可对相机的仰角以及偏航旋转速度加以限制。通过限制相机与世界 Up 轴之间的平行关系（即使目标对象位于当前相机的垂直上方或下方处），可有效地避免诸多数值计算问题。另外，相机前视向量与世界 Up 轴之间的最小夹角亦不宜过大，或许 1° 左右为最佳。因此，方向插值计算应支持此类限制条件，针对于此，可分别在仰角以及偏航角方向上进行插值计算。

插值选择方案

若插值数据类型以及方案已知，针对某些特定场合，应尝试选择问题的最佳匹配方案。

首先，这将引出第一个与插值计算相关的决策问题：所提供的数据值是否可体现实际有效值，并在某一时间间隔内由插值函数进行复制？若答案是肯定的，则该值通常称作关键值或在动画中称之为关键帧。除此之外，该数据值是否可视为控制值，并用于影响生成插值数据值（或近似值）的相关函数？

其次，插值计算的性能问题亦不容小视，例如，基于多级层次结构模型的全动画系统，其插值方案与相机移动插值往往截然不同。降低精确度常会减少处理器的插值计算开销，尽管此处并未采纳此类建议。

第三个问题与插值结果的平滑性（或称作连续性）相关，与角色动画相比，相机移动以及方向的平滑性则具有更为重要的地位，皆因此类平滑性特征将影响到整体场景画面的渲染效果，而非某一模型对象。

与插值计算相关的问题还包括，是否有必要针对各关键数据值存储特定的视见间隔数据？抑或是否可在关键数据之间采用均一的时间值？后者则可降低所需数据量，但这将会产生插值率问题，具体内容稍后解释。

最后，插值类型接口问题以及所需数据量也应予以考虑。需要注意的是，不同的插值方案通常具有不同的计算需求，例如，Hermite 曲线往往在各关键值处需要使用到切线数据。另外，针对所使用的插值类型，是否采用适宜的编辑工具也可视为一类决定因素。例如，与分段式线性编辑器相比，Hermite 或其他样条曲线往往需要更为复杂的编辑器以及表现方式。

线性插值

线性插值可视为一类最为简单的插值方式，该过程仅需要 3 个参数，即源值、目标值以及对应插值方案。其中，该插值方案用于定义计算过程中的各数值结果。

源值

顾名思义，源值定义了初始状态下的插值数据。此处，读者应了解与源值变化率相关的某些背景知识，尽管针对线性插值计算而言这并非必需。针对线性插值计算，源值可视为常量值，并在插值计

算的构建阶段加以使用。

目标值

目标值则定义为插值过程中的最终值，该值可设置为常量值，并可对插值过程实现良好的控制——这可视为一类最为简单的情况。然而，多数时候，目标值往往会在一段时间内发生变化，并会产生不连续性等一系列问题。考虑到当前任务之一是令插值数据与目标值实现良好的匹配，因而需要对目标值的变化率加以考察，而非该值自身。当插值数据的变化率与插值结束阶段的目标值变化率无法实现较好的匹配时，即会产生不连续问题，类似的问题同样也会出现于插值过程的开始阶段。

插值算法

插值算法体现了源值与目标值之间的插值数据变化方式，这里，首先需要考察的问题是是否有必要在固定时间值内实施插值计算，抑或插值率是否存在变化，这在很大程度上取决于插值量的特征。需要注意的是，位置插值过程与颜色空间或 FOV 插值方式存在着较大的差异；除此之外，还需对目标值的常量或变量等特征加以考察。

另外，源值与目标值之间的简单的插值计算往往对变化率不予考虑，并将其统一视为常量，此类方案的实现方式通常较为直观。相对简单的插值计算其高效性不言而喻，但往往会在插值过程的开始和结束阶段产生不连续现象，当处理某些快速变化的数据值时尤是如此，例如相机的方向或移动行为。当目标值定义为常量时，该值的非线性插值过程将更为高效，尤其是处理某些与相机相关的数据值时，如方向数据。

分段式插值

某些情况下，插值数据值并非仅限于源值以及目标值这两个数据之上，然而，与线性插值相比，插值数据量的增加并未降低操作过程的简便性。出于连续性考量，一种方案是将值序列视为彼此无关的插值线段。随后，可对各线段采取彼此独立的线性方式实施插值计算。为了计算某一线段中的时间因子，有必要针对各数据值存储对应的时间间隔值。通过分析时间值数组（可采用二分查找方式），则可定义相关显示以及（该线段内的）标准化时间因子，具体过程如下所示：

```
bool foundTime(false);
for (int32 i = 0; i < valueTimes.size()–1; + + i)
{
  // a linear search, binary search would be
  // preferred
  if (valueTimes[i + 1] > time) // assumes sorted
  {
     float timeDelta = valueTimes[i + 1] –valueTimes[i];
     timeFactor = (time – valueTimes [i]) / timeDelta;
     segment = i;
     source = values[i];
     destination = values[i + 1];
```

```
        foundTime = true; // we have found the right time
        break;
    }
}
if (foundTime)
        return Interpolate(source, destination,timeFactor);
else
...
```

另外，此处假设各数据值之间均采用均匀时间间隔值，这可有效地减少数据，但问题亦随之而来，根据各条线段开始和结束阶段的值差异性，线段间的插值率将发生显著变化。这不仅会导致非均匀的插值率，根据插值量的特征，线段边界处还会出现明显的不连续现象。针对典型的相机数据值，这将在穿越边界时产生跳动现象或显著的方向变化，对应代码如下所示：

```
// constant segment time interpolation
segment = time / kSegmentTime;
source = values[segment];
destination = values[segment + 1];
timeFactor = (time – (segment * kSegmentTime) ) / kSegmentTime;
return Interpolate(source, destination, timeFactor);
```

插值方案

在某种程度上，插值方案的选择可通过环境的特定因素以及插值特征加以确定，例如，相机的位置以及方向可通过非线性插值方式获取较好的计算结果，其过程的开始和结束阶段均呈现出较为平滑的加速和减速状态。

当对相关方案进行评估时，首先考察插值过程是否应采用常量时间值，抑或根据插值数据的差异性使用变化值（以及此类插值数据的变化率）。后者有时亦称作基于速度的插值计算，由于插值过程的持续时间往往处于变化中，因而该方案较少使用。当前过程仅考察常量时间值的插值计算，并使用采样间的固定时间间隔。

线性时间插值

顾名思义，如果源值以及目标值根据时间间隔值进行分布，则线性插值计算可视为绘制于值间的直线。若 t 为当前标准化插值因子，则可按下列方式计算该直线上的位置数据：

```
Interpolated value = source + ((destination – source) * t)
```

其中，在源值处 t=0，且在目标值处 t = 1。如果源值以及目标值处的时间值不为 0、1，则需将其重新映射至下列范围：

```
Total time = destination time - source time
t = (desired time - source time) / Total time
```

这里，考虑到浮点误差将导致当前数值与期望值之间存在少许差异，因而应确保时间因子位于 0～1 范围内。另外，插值结果应与目标值进行比较，而非源值。针对于此，前述公式可修改为：

```
Interpolated value = destination - ((destination - source) * (1 - t))
```

线性插值计算适用于多种数据类型，尽管如此，该方式依然存在某些缺陷，具体插值操作将在“球体线性插值计算”一节中加以讨论。

参数函数

若不采用前述直线近似方案，则可尝试使用各类参数函数，并在既定时间间隔内传递源值和目标值。类似于上述线性时间插值方案，可根据流逝时间量（并与初始值和目标值的时间间隔进行比较）计算标准化的插值数据。随后，可利用参数函数将其映射至不同的非线性标准化值，进而再次对原标准化值实现参数化，该新值可用于确定插值数据的值域范围。

这里，参数函数需返回标准化值且无其他特定要求，这一类函数通常称作 ease 函数。

球体线性插值计算

球体线性插值简称 Slerp，该插值方案对应于前述情形，即线性角度插值无法获取线性角速度。角度插值计算通常会对单位球体上两位置间的弦长实施插值操作；针对该问题，Slerp 则根据同一位置间的表面弧长移动插值点，且位于两位置以及中心点所构成的平面内。

Slerp 操作可极大地改善插值质量，但通常会产生较大的计算开销，即使小角度下的线性插值近似方案亦是如此。然而，当采用分段式插值方案时，其结果将产生较为明显的不连续现象（即跨越线段边界）。这里，再次考察单位球体并得出以下结论：

各分段式插值过程彼此不同，即使同一线段内部呈现为平滑状态。

潜在插值问题

相机插值计算以及相关操控属性涉及了大量的数值计算以及美学问题，相关问题均可通过绑定于相机之上的渲染视口加以展现，因此，相机属性应对此进行适当的调整（至少针对处于激活状态下的相机）。需要说明的是，在某些情况下，插值计算并非总是最佳处理方案。在场面较为激烈的情况下，鉴于快速移动以及方向变化可有效地转移玩家的注意力，因而建议采用相机的瞬间移动以及重定向操作。

同时，关于插值问题，还可从以下两个主要方面加以考察，即美学问题和数值问题。

美学问题

相机插值计算受限于诸多潜在的美学需求条件，相关内容并不会对插值计算产生致命的影响，但却会产生某些负面的图形效果，并降低游戏场景画面和谐感。若关卡设计人员或美工师对相机系统加以设计，则多数问题均可得以避免。这也意味着，若设计人员具有相机插值方面的相关背景知识，则

可有效地消除游戏场景中的某些问题。考虑到游戏体验的动态特征（如游戏对象的移动行为），某些具体问题也应予以考虑。例如，某些时候，相关方法可能会提前结束插值过程，抑或根本无法启动插值计算。

这一类问题均会使相机跳转至目标相机处。

下列内容列举了某些较为常见的美学问题。

几何体相交问题

有时，插值相机将会穿越游戏对象或当前场景；类似地，相机与渲染几何体之间的距离可能过于接近，并导致相机的近剪裁面与该几何体相交。该问题应尽量避免，其原因在于，此时相机将向玩家展现游戏的内部工作结构，进而破坏当前角色与场景间的美感。除此之外，考虑到几何体相交通常易于处理，因而相关解决方案不必过于专业。

一类常见的方案将对当前场景世界进行考察，进而判断相机的当前移动或外来移动是否会产生碰撞或内部相交问题。当然，此处内容主要涉及近剪裁面相交问题而非碰撞空间体。一般地，该方案可通过一条或多条投射光线加以实现。具体而言，一条光线可沿相机移动方向进行投射，另一条光线则沿相机的前视向量进行投射（考虑到相机的移动方向与视见方向并非一致）。

移动投射光线通常将相机视为一点，且从当前位置扩展至新位置。其中，新位置可通过插值计算获取。如果投射光线与碰撞几何体相交，则移动行为很可能处于未完成状态，且需要对此进行处理以避免产生相交问题。

沿前视向量的投射光线则始于相机新位置处（根据移动光线投射，该位置可能稍有变化），其扩展距离稍大于距相机的近剪裁面距离。另外，投影光线将与碰撞几何体或渲染几何体进行比较，进而判断是否存在相交现象。当然，仅当碰撞几何体完全包围渲染几何体表面时，碰撞几何体方案才方显其应用价值。渲染几何体则更具视觉精确性，但其计算开销往往令人望而止步。如不考虑几何体的测试过程，针对相机视锥体与渲染几何体之间的全测试过程，沿视见方向上的单一光线可视为一类与此对应的优化方案。然而，视锥体顶角依然可能与渲染几何体相交，尽管此时投射光线并未与其相交，该情形通常与 FOV 相关。这里，可适当地调整光线投射的延伸距离（超出近剪裁面少许）即可解决以上顶角相交问题。

需要注意的是，无须沿相机方向执行光线投射，使用源于渲染深度缓冲区（或 Z 缓冲区）中心位置处的一个或多个数据值已然足够。该方案不仅可降低处理器开销，考虑到相关操作围绕场景渲染几何体执行，因而其精度大大提升。除此之外，如果对象的碰撞问题未予考虑，则深度缓冲区中的部分内容可在对象渲染之前进行检测。通过此方式获取的深度信息通常源于游戏场景的前次渲染操作，且相机可能与渲染几何体之间过于接近，因而需要执行较大范围的距离检测。

通常情况下，光线投射方向应从新位置逆向至前次相机位置处，其原因稍后加以解释。

在预定义移动方案中，如样条路径，整条路径可在插值移动开始之前加以验证，必要时，甚至可省略该插值过程。然而，由于游戏对象可能会在插值过程中移至相机路径内，因而预置方案并不能满足全部要求。

一旦发现潜在碰撞问题，则存在多种可选方案，具体内容如下所示：

☑ 结束插值过程并调整至目标相机处。

☑ 跳转至（远离障碍物）路径上的某一位置。

☑ 穿越几何体并忍受某些图形缺陷。

不连续性

线性插值并不与插值属性的变化率匹配，且仅采用了源值和目标值。同时，既定插值过程中的目标值很可能在一段时间内发生包变化，如目标相机位置。由于步长在插值过程中不断改变，因而较为直接的位置线性插值方案无法确保移动的平滑性特征（实际上，该方案可能导致相机远离目标位置）。另外，当未采用缓入/缓出功能时，插值计算的启动以及结束动作则稍显“僵硬”。该问题多通过插值相机与目标相机之间的速度差异，或目标属性处的振荡现象加以体现。一类处理方案是，与插值属性的变化率实现良好的匹配，而非自身数据值。

插值噪声

鉴于相机的位置和方向均处于变化状态中，因而整体渲染场景受限于插值过程所引入的误差。因此，应考虑以相对平滑的方式调整位置和方向等数据值；同时，还应尽量避免浮点误差所引起的方向遮挡现象。针对于此，可尝试采用高通过滤器以使插值结果收敛于目标值处。另外，由于步长误差将导致目标值的过冲现象，因而基于 Epsilon 值测试的插值方案依然会产生问题。

目标对象取景

如前所述，简单、直接的方向线性插值方案无法保证目标对象位于视见范围内，即使源方向和内部方向直接指向目标对象时亦是如此。当目标对象快速移动，抑或目标对象与相机之间的距离较小时，该问题则十分明显。此类问题可通过以下方式加以解决，即在插值前期对插值数据实施加权运算，以使目标对象保持可见。目标位置插值计算通常会遭遇到这一类问题。

大幅度的方向变化

即使方向变化呈现为平滑状态时，若角度距离较大，则观察者依然会失去原有的方向感。限制方向插值计算过程中的角速度可视为一类潜在处理方案，尽管这可能会导致目标对象脱离当前视锥体。当出现快速的旋转变化时，控制参考帧问题也应予以考虑，读者可参考第 8 章以获取更多内容。若初始角距较大，则可直接针对目标相机实施跳转操作，进而表面玩家失去方向感。

较大的插值距离

如果源值和目标值之间的插值距离较大，则可先期执行插值计算，并与随后实施跳转操作。当插值过程持续时间较短时尤是如此，这将导致相机在场景内的高速移动。需要说明的是，在应用过程中，此类实际方案常与具体的游戏环境相关，且仅出现小范围的方向变化时，该方案方为有效。

数值问题

插值计算中的数值问题多与浮点表达方式或奇点造成的限制条件相关，具体问题包括以下几点：

共线方向（或反向）

四元数插值或其他向量数值运算需测试这一类情况，以防止出现异常。针对于反向共线，则旋转方向呈现为任意状态，相关计算将根据之前的状态信息加以执行（例如，选择与前次插值过程相同的方向）。

位置一致性

向量位置之间的减法可生成 0 长度值（或较小量），这将导致除 0 错误或其他问题，当执行平方根计算时尤是如此。

浮点误差

由于浮点数无法反映出对应数字的真实值，因而在进行数值比较计算时往往会出现问题。因此，当判断相关数值是否位于 Epsilon 范围内，并以此执行插值结束条件的测试时，浮点误差常会导致测试失效；当相机位置远离世界原点时，该问题则变得尤为明显。关于浮点误差问题，读者可参考[Ericson05]或[Crenshaw00]以获取更多内容。

插值中断

当相机插值结果出现于某一新插值位置时，将会触发某些外部游戏事件，该情形在游戏体验过程中十分常见。此时，常见的解决方案则根据插值相机的当前属性（位置、方向和 FOV 等）启动新一轮的插值过程，其结果尚可令人接受，但具体情况还取决于源插值目标属性与新目标属性之间的差异。通常，新目标相机方向与源目标方向之间存在较大差异，这一类差异特征使得渲染场景产生明显的变化，并伴以某些负面效果。类似地，这将导致相机插值属性产生不连续现象。该问题可通过匹配属性的插值率加以解决，而非简单的插值数据。

另一种与插值中断相关的问题是，相机的移动行为可能会违背 30° 规则。某些时候，可采用相应的视觉元素以强调相机的移动行为，如屏闪效果，该效果可通过屏幕过滤机制加以实现，即在不遮挡游戏场景画面的前提下调整渲染画面的外观。例如，系列游戏 Metroid Prime 即采用了扫描线过滤器，其中，位于显示视见范围内的隔行水平像素均通过 Alpha 透明度效果进行调整。

切换过程

相机间的切换操作则采用了多种前述插值技术，然而，切换操作应精心加以设计，以防止相关过程产生意外变数（相对于常规插值计算而言）。

切换过程中的位置数据

相机在切换过程中的移动行为应以平滑方式进行，且与玩家移动无关。同时，针对改变玩家移动或重定向方式的某些外力因素，相机也应对此加以适当考察。例如，若相机沿生成样条路径移动（即从第三人称位置至第一人称位置），则路径方向以及位置数据应与玩家方向保持一致，即使玩家角色正在执行重定向操作。正常情况下，基于期望相机移动行为的快速变化通常不可取，此时，观察者的主参考点（即玩家角色）将保持不变，这将使问题得到一定程度的缓解。

切换过程中的方向数据

在相机间的切换操作过程中，存在两种主要方案用以控制插值相机的方向数据。若源相机和目标相机采取重定向操作，则可使用相对简单的线性插值或球体线性插值方案。此处，使用四元数可有效地解决奇点问题。

然而，若相机在切换过程中涉及重定向操作，则简单的插值方案往往无法满足任务要求，原因之一是，相机实施转向操作，进而方向间的夹角可导致当前插值量将切换相机远离目标方向。针对于此，较好的解决方案是在连续更新操作中不断逼近目标属性值，而不考虑目标相机的变化，为了确保该过程顺利执行，一种方法是计算目标值与当前值之间的剩余量（此处为角度值）。若最新插值数据大于当前值，则该数据将被忽略。另外，插值过程还可设计为采用固定时间量，且暂不考虑源相机和目标相机的移动行为以及重定向操作，此类情形也颇为常见。

相机数学问题

若读者了解数字的浮点工作机制，则会对该格式的误差问题持有深刻的见解，毕竟，浮点格式仅是一类近似表达方案。该问题的表现形式相对微妙且一言难尽。针对游戏程序设计中的其他因素，健壮的浮点处理方案使得相机系统的行为方式令人更加信服。读者可参考[Ericson05]以及[Crenshaw00]以获取更为全面的浮点解决方案；除此之外，读者还可阅读针对当前目标系统的处理器操作手册，以获取详细的浮点表达信息。

下面将对某些较为常见的问题进行简单的介绍。

浮点精确度

相机系统通常采用单精度浮点数；双精度浮点数则可在一定程度上提高精确度，但其计算开销也将随之上升。针对长距离数据，精确度问题则显得异常重要。考虑到浮点数有限的表达范围，随着数值不断变大，该过程将不可避免地产生精度损失问题。

使用 Epsilon 值

如[Ericson05]中所述，许多游戏并未正确地处理 Epsilon 值问题，该值多用于确定浮点数的接近程度，针对相机而言，该值常用于确定方向以及位置等相关内容的相等关系。

一类简单的方案则根据精确度的可接受程度将 Epsilon 值定义为固定值，然而，正如 Ericson 所指出的那样，Epsilon 值应取决于比较值的范围。而且，考虑到浮点数在实数轴上的间隔呈非线性状态分布，因而这也将对 Epsilon 值的精确度造成一定的影响。

例如，读者很容易陷入一种假设模式，即等值判断总会生成正确的结果；或者，固定 Epsilon 值即可确保浮点比较操作得以正确执行。针对于底层浮点表达方式，[Ericson05]对此类问题进行了较为透

彻的分析；另外，针对浮点值的比较操作，该专著也给出了比例 Epsilon 值的正确用法以及对应的实现方案。

简而言之，当需要执行浮点比较操作时，应适时地调用 close- Enough()函数，进而避免使用直接的等式比较机制以及 Epsilon 值检测操作。这一类集中函数使用起来较为方便——函数将根据具体用途适当地调整 Epsilon 值。

编译器偏差

针对显式的浮点除法操作，编译器所提供的实数精度其可靠性有待商榷，例如：

```
static const float kDivisor(10.0f);
static const float kMultiplicand(10.0f);
static const float kFoo = (1.0f / kDivisor) *
kMultiplicand;
```

若 kDivisor 和 kMultiplicand 定义为等值，则 kFoo 的结果值应为 1.0f。但考虑到处理器的除法计算方式，kFoo 的正确值无法得到有效的保障。在实际操作过程中，根据当前编译器的具体情况，kFoo 值可在 0.9～1.0 之间变化。

硬件 FPU 偏差

针对多目标平台任务，应确保硬件与浮点计算之间的一致性，通常情况下，这一类问题可由编译库加以解决；健壮的浮点设计以及实现方案应考虑到可能出现的任何细小偏差。相机的标准化检测操作则多取决于具体的需求条件，如有效的方向向量以及向量间的差运算。

向量标准化操作

多数向量操作都会使用到标准化或单位向量，进而返回具有一定意义的计算结果。同时，向量的标准化计算还可用于判断视见计算是否可以正确地执行，亦即，该计算可计算空间内两个 3D 位置间的接近程度。当然，较好的方案则是计算数据的偏移量。

矩阵累积浮点误差

待使用旋转矩阵后，可针对各个基向量实施矩阵的标准化操作，进而消除一段时间内所引入的浮点误差。

定期修复数值

当了解了与相机数值计算相关的某些潜在问题后，即可采取对应措施以防止此类问题对相机系统

产生负面效应。当执行某一特定操作时，某些问题需得到即刻处理；而另一些问题（与相机方向变化相关）则可通过一段时间内的分摊机制加以处理。

在各更新循环的结束阶段且渲染操作执行之前，针对当前游戏环境，建议对当前转换矩阵的有效状态进行验证。这里，某些检测操作可归类于非产品创建过程；但针对大部分内容，则应设置适当的检测条件以处理某些致命错误。典型的帧检测包括以下内容：

- ☑ 转换矩阵分量不应包含不适宜的浮点值，例如 NaN，该问题可通过恢复前次有效缓存矩阵加以解决。此处，若当前位置数据正确，建议仅恢复旋转分量即可。
- ☑ 构成的转换矩阵的独立分量应定义为单位向量。
- ☑ 针对当前游戏，Up 向量的方向应保持适宜状态；通常情况下，应确保 Up 向量与世界 Up 向量指向同一方向。
- ☑ 必要时，可移除围绕前视向量的旋转操作。
- ☑ 必要时，应限制相机的快速重定向操作。

若矩阵无法修复，应采取上一帧所使用的矩阵数据，进而获取有效的视见内容。然而，新的位移数据其使用方式可独立于相机的方向数据。

最后，应适时地对相机矩阵进行标准化操作，以消除连续矩阵乘法操作所产生的累积效应。

第 11 章　相机实现方案

本章将考察实时应用程序中相机系统的实现方案，并分为以下两个部分。首先将讨论相对复杂的结构问题，其中包括子系统的划分以及针对类定义和数据结构的相关建议。其次，后续内容还将阐述相机系统与游戏引擎之间的整合方式，以及开发工具和调试功能。另外，相机系统多采用 C++语言加以实现，尽管多数算法也可采用其他语言进行描述。

在本章开始之前，建议读者简要地回顾一下第 1 章中的相关内容，其中包括标准的游戏循环操作。针对于实时相机系统，第 5 章也曾探讨了诸多游戏设计决策方案，相关内容对于相机系统的设计以及实现过程均提供了有益的帮助。

当定义相机系统时，相关功能应满足对应的游戏需求，而非错综复杂的概念符号。需要注意的是，本章所包含的针对于类和数据结构的相关建议并不具有普遍性。针对于既定项目，公共相机系统的数据元素仅限于该项目自身，针对于组件式相机系统的实现过程，该方式可大大简化设计过程，具体内容稍后讲解。

本章的目的并不仅限于提供成品相机系统，本章将针对相关设计过程提供良好的建议；如果读者对相机系统的复杂度缺乏应有的了解，则本章将引领读者直击事实的真相。

同时，相机系统还依赖于其他外部系统，其中包括配置检测系统、控制器输入系统、场景渲染系统、音频管理系统等。针对上述系统与相机系统之间的交互方式，在相机系统的实现过程中，还应关注交互过程中的依赖关系以及其他一些细节问题。无论如何，相机系统具备良好的自包含特性且理应作为共享库加以实现。针对于此，可采用数据驱动方案以定义相机的行为，而非针对某一特定游戏的基于硬编码的相机方案。

游戏引擎体系结构

第 1 章中曾讨论了典型的游戏引擎及其与相机系统之间的工作方式，其中，存在多个与相机系统相关的游戏引擎组件。相关内容如下所示。

游戏更新循环结构

游戏相机多依赖于其他游戏对象的更新操作，进而可正确地处理其移动以及方向行为。因此，多数相机逻辑需在逻辑更新循环结束阶段加以执行，毕竟，在渲染任务执行之前，多数与移动相关的问题已处理完毕。通过该方式，相机逻辑将与游戏对象的近期状态发生关联。在第 1 章中，典型的游戏子系统执行顺序如下：

☑ 处理输入数据。

- ☑ 处理游戏对象的预逻辑。
- ☑ 处理游戏对象的主逻辑（如 AI 对象）。
- ☑ 移动对象并解决碰撞问题。
- ☑ 处理全部相机逻辑。
- ☑ 处理对象的后相机逻辑（如相机位置、方向等内容的相关性）。
- ☑ 处理全部视口逻辑。
- ☑ 针对各视口渲染当前场景。

游戏系统管理器

如第 1 章所述，管理器可视为游戏架构中较为常见的功能系统整合方案，各组件分别对应于独立的游戏（或引擎）逻辑。典型的游戏引擎通常包含以下系统：

- ☑ 相机管理器。相机管理器可视为本书的重点内容，这一类管理器负责跟踪全部相机行为、将输入数据传递至相机系统并确保更新操作得以正确的执行。
- ☑ 对象管理器。对象管理器用于处理游戏对象间的传递机制、游戏对象逻辑顺序以及逻辑表内的对象分组机制（如 AI 对象以及门对象等）。对象的空间顺序以及游戏体验顺序应避免搜索激活对象时所产生的计算开销（以及维护表数据时所产生的开销）。这里，根据对象属性（如 AI 对象）实施排序操作，但应尽量避免多次执行相关逻辑内容，其原因在于，同一对象可能位于多个表项中。游戏对象的逻辑执行顺序通常与内存中的对象存储状态以及标识符的独特性相关；同时，对象间的依赖关系也需要加以显式排序，进而可对其关系实施正确的维护。
- ☑ 消息管理器。在某一时刻，脚本事件需要将消息传递至对应的脚本对象，并使该对象获取相应的视见内容。该机制也可用于对象间的交互过程。
- ☑ 音频管理器。这里，可通过脚本系统控制音频效果的初始化工作。除此之外，还可根据游戏场景中的位置设定立体声效果或环绕声效。这一类声音效果需要结合当前活动相机的位置和方向信息，以使相机有效地对音频管理器实施更新操作。另外，当采用多视口时，还应进一步考察立体声以及环绕声效的处理方式。单玩家游戏常采用线性立体声映射，其中，立体声场源于相机前视向量与相机至 2D 空间声音位置向量之间的夹角。这里，采用 2D 空间的原因是，人类的声音感知系统通常无法辨识纵向位置的差异性；同时，音频系统通常不包含声场内的纵向定位信息。如果相机很少出现倒置状态（如某些飞行模拟器），则可忽略围绕前视向量的旋转操作。
- ☑ 输入管理器（IM）。向全部游戏对象传递输入消息通常需要输入管理器的支持，同时还可能包括活动相机以及视口等对象。典型的实现方式则是令 IM 向各主游戏系统提供输入数据，当然也包括相机管理器。随后，相机管理器将输入数据传递至各个处于活动状态的相机，并检测相机是否从特定的控制器中查找输入数据。输入控制器可与物理控制器输入以及处理后的数据结合使用。此类实现方案应包含控制管理器，并以游戏定义方式对物理控制器按钮、按键操作以及游戏杆数据值实施映射。当然，该过程还包括按键的抖动消除、参考帧的维护等多项内容。当采取适当抽象后，控制管理器可向玩家提供多种控制排列方案，以供玩家选择。

输入管理器多与底层硬件相关，并负责处理真实的硬件值渲染操作（提供系统库直接与硬件发生关联）。尽管控制管理器工作于较高级别之上，但它依然提供底层输入数据的访问能力（仅限于控制器按钮/值级别）。

时间间隔

现代游戏体系结构通常会将游戏的图形更新操作从相应的逻辑更新中分离开来，这也使得 CPU 处理（包括多核处理器）与 GPU 渲染操作之间的并行操作成为可能。鉴于游戏逻辑的更新过程与前次场景的硬件渲染过程同步执行，因而系统整体性能将得到很大的改善。游戏逻辑运行时的更新率可保持恒定不变；当然，适当的变化也在允许之列——最终，相机系统更新操作之间的时间量其一致性将无法得到保证。自然状态下，相机应可与各种不同的更新率协同工作，因此，全部移动行为以及基于时间的属性不应针对特定的更新率进行硬编码。相反，上述相关属性值应根据单位时间（秒）内的变化率加以计算，并按比例调整至距前一次逻辑更新操作的流逝时间值。此处，流逝的时间值常称作时间间隔。

在实际操作过程中，移除基于更新率的依赖关系则具有诸多优点。首先，不同电视标准之间的转换操作将得到大大的简化；其次，在游戏逻辑执行过程中，CPU 使用状态的突然变化所导致的短暂的帧速率变化基本不会对当前相机系统产生任何影响。如果游戏需要对录制后的前期游戏视频加以播放，则应存储与各逻辑循环操作相关的时间间隔值。更新操作间的流逝时间变化将导致游戏逻辑与原始录制视频序列不符，需要注意的是，针对于处理可变流逝时间值的相关逻辑内容，应可有效地处理 0 时间间隔值（或接近于 0 值）这一特殊情况。由于时间间隔值多出现于除数位置上，因而应避免出现除 0 以及溢出等错误。

输入处理机制

多数相机均需要从主游戏逻辑中向其传递控制器输入值，该过程通常由相机管理器进行处理。某些时候，特定相机所使用的输入值可能与基于当前相机的控制器无关，调试功能常需要使用到不同的控制器（例如，源于网络的输入值而非本地控制器，或本地备用控制器）；抑或，游戏演示回放或最近玩家行为将从其他资源处提供相应的输入数据。

根据对应的游戏需求条件，尽管其他游戏通常会接收经过滤后的输入值，而游戏相机则仅接收原始数据，并在必要时将其提供至其他过滤机制。除此之外，该过程亦可在相机管理器中加以处理，进而确保此类逻辑在全部游戏对象之后运行。

相机系统体系结构

如果不考虑个体游戏相机的特定表现风格，相机系统体系结构的设计过程将包含诸多公共元素。因此，可对相机系统层次结构藉以适当的抽象，并在渲染处理之前暂且忽略其表现风格。当然，抽象的程度往往取决于特定的游戏需求，但系统的复用性应始终贯穿于其中。

从概念上讲，表现风格的易辨性将非常有用，此处场景视见内容的投影将具有较为明显的外观差异。出于简洁性考虑，这里仅采用 2D 或 3D 相机，从技术角度上讲，二者之间的差异亦仅是投影转换矩阵不同而已。关于相机表现风格等问题，读者可参考第 2 章以获取更多信息。

相对于相机系统，其他外部管理器则负责处理场景渲染以及相关内容的显示工作。另外，相机管理器应包含访问功能，以在必要时提供相应的场景信息。

视口管理器

针对于多玩家游戏，视口管理器（VM）体现了最高层次的相机系统抽象，这一点十分重要。这里，视口管理器的主要任务是将全部场景渲染信息整合至部分输出表面中，输出表面可以是处于激活状态下的帧缓冲区，或应用于游戏场景表面的游戏纹理。VM 应具备相关功能项，以使渲染引擎可正确地显示当前场景（如视锥体信息、投影类型、颜色深度和输出尺寸等），并可实现不同视口之间的切换操作。

针对于各个视口，VM 将处理控制器输入、渲染、遮挡、缓冲区管理、宽高比调整等多项内容，并在数值传递至（特定视口中的）对象之前对其实施过滤或调整工作。另外，其他游戏系统也可通过 VM 查询相机系统，以获取与当前激活态相机相关的信息以及属性。

根据显示媒介所受的物理限制，VM 应可在不同方式下工作，相关结构如下所示：

```
class CViewportManager
{
public:
    typedef int32 TViewportHandle;
    TViewportHandle const CreateViewport(EViewportType);
    void DeleteViewport(TViewportHandle const viewport);
    void Update(float const deltaTime);
    void ProcessInput(CInput const & input);
    CViewport const & GetViewport(TViewportHandle const handle) const;
private:
}
```

当显示结构定义完毕后，则应针对特定游戏视野中的投影方式加以考察。

VM 涵盖了显示渲染场景时的诸多游戏元素，这里，VM 可视为一类游戏对象（亦即，该对象继承于基本游戏对象类，如 CGameEntity）并需要在每帧中进行更新操作，但无须在游戏场景中显式地设置物理位置（尽管 CActor 类包含了相应的物理描述）。

这里的问题是，VM 应提供何种功能？下列各项建议显示了基于期望特征集的潜在内容表项：

- ☑ 视口访问器。内部视口表中的视口对象应可提供有效的视口句柄加以访问。通常情况下，针对外部调用函数，应仅提供 const 访问。
- ☑ 全视口渲染。遍历内部视口表并针对全部处于活动状态下的视口调用 Render()函数，该过程可能仅生成渲染信息表（如视见转换、视域、宽高比等）并传递至主游戏渲染引擎中。
- ☑ 输入控制器（某些视口可能需要访问控制器输入值）。遍历内部视口表并将控制器输入传递至基于活动窗口的 InputHandler()函数。
- ☑ 创建（构建）新视口。在创建一个新视口时，函数应返回基于该视口的唯一句柄，该句柄随后被缓存并用于区分视口。

- ☑ 激活或禁用当前视口。该功能可通过有效句柄对现有视口加以控制，这一类功能项通常仅用于设定内部状态，并定义视口当前的操作状态，即执行更新操作抑或渲染操作。
- ☑ 删除现有视口。再次强调，可从视口管理器中移除基于有效句柄的视口对象，并将其内存空间返还于当前系统。
- ☑ 视口间的切换（变形）。当定义了两个有效句柄后，即可对渲染切换实施初始化操作，通常情况下，这将确定切换操作类型（稍后解释）以及对应的持续时间。
- ☑ 调整视口属性。这里，应适当地提供动态确定视口位置或尺寸的能力（该操作一般通过有效的视口句柄加以处理）；同时，该操作可采用以下两种形式：即刻变换或一段时期内的插值变换。针对于后者，可设置 SetDimensions()函数中的可选参数加以实现。
- ☑ 针对视口设置活动相机。一般地，可将相机 ID（句柄）绑定于视口之上，该相机 ID 可从相机管理器中获取视见转换矩阵。因此，多视口可使用同一相机对象，且游戏相机无须了解具体的渲染处理过程。

视口

各个视口在其显示空间范围内控制着渲染操作的方方面面，因此，视口通常涵盖与相机、关联控制器输入、表面位置及尺寸、渲染模式等内容相关的信息；必要时，视口还可包含渲染表面相关的信息，进而支持具有可变显示能力的多重输出表面类型。需要注意的是，视口将采用单一相机集，标识符 TCameraId 可通过相机管理器（CM）访问相机信息。这里，视口将包含丰富的信息以对当前场景实施渲染操作。

视口切换

这里，VM 负责控制不同视口间的切换操作，且多出现于游戏开始前的菜单项遍历操作、游戏的暂停操作等。某些时候，游戏体验过程中可能还会包含画中画功能，以向玩家展现迥然不同的游戏场景，该情形多出现于机动设备模拟器中。有时，分离的视口还可用于显示主场景画面的近距离特写内容（即放大操作）；另外，通过视口在同一输出设备上（通常为显示器）显示多场景画面亦十分常见；有时，多个视口还可采取叠加的方式加以显示。

切换操作的执行方式亦多种多样，且与视口是否执行同步更新操作无关。下列内容显示了较为常见的效果示例，读者也可参考相关电影技术专著以获取更多内容。

- ☑ 跳转操作。跳转操作可视为一类最为简单的切换技术，此情形多发生于新视口即刻替代原视口时，且分离场景画面之间不存在任何叠加效果。此时，视口的尺寸彼此相仿且在显示设备中占据同一位置。针对于全屏视口，只需要对相机进行适当跳转即可，除非投影或宽高比亦发生变化。
- ☑ 移除换景。移除换景是一类较为常见的影视技术，其中，当前视口内容逐步被清除，并以某一颜色所替换（通常为黑色，有时也采用白色）。另外，清除方向多呈现为轴对齐方式、对角线方式、菱形抑或圆形等方式，在切换过程中，原视口清除部分将被新视口的相关内容所替代。除此之外，原始内容或新内容还可在切换过程中实施更新操作（取决于 CPU 条件）。通

过采用源、目标场景之间的混合函数，切换过程中较为明显的拖曳边则可得到不同程度的柔化，进而使得开始和结束阶段之间呈现为更加自然的过渡效果。

- ☑ 交叉渐变。这里，可在原始视口内容与新视口内容之间实施混合操作，进而替代原视口中的画面内容，该过程可通过淡入新内容、淡出原内容的方式加以实现。出于高效性考量，视口间应保持同一尺寸，并占据相同的屏幕空间。由于两个视口需执行同步渲染操作，因此此类操作通常会产生较大的计算开销。针对于此，该问题可通过下列方式加以解决，即采用捕捉后的源场景画面副本对新视口实施更新操作，当然，这会在一定程度上降低原有的美学特征。
- ☑ 变形。此处，存在多种方案可对变形操作实施处理，若视口间具有相同的物理形状以及宽高比，则可对位置以及尺寸数据采取插值计算。其中，插值计算过程亦不一而同，例如，视口的线性移动、视口以 2D 或 3D 方式进行旋转等。除此之外，视口还可分解为多边形（如彼此镶嵌的多个三角形）或高阶表面，并实施相应的混合操作。其中，可对多边形顶点或纹理坐标进行插值计算。此类多边形网格可通过多种方式加以操控，如旋转相关、视口的卷曲/展开效果等。

针对于上述切换效果，应在切换过程中定义相应的时间量，该时间值可随后用于控制混合程度等操作。例如，2D 样条控制，或者分段式 Hermite 曲线（可将时间值映射为其他数据，如 Alpha 透明度）。

渲染管理器

渲染管理器可从视口、相机以及游戏系统的各类状态中获取数据，并将某一场景画面渲染至表面上。除此之外，该管理器还包括以下功能，即移除位于当前视锥体之外的对象几何体、确定正确的渲染顺序以及使用后过滤效果。

相机管理器

针对游戏内的全部相机对象，CM 饰演了一类控制器角色；然而，必要时，还可采用多个相机管理器（针对于多玩家游戏而言，每一个视口均对应于一个相机管理器）。同时，CM 也可视为基于视口管理器的一个借口，进而访问与活动相机相关的全部信息。必要时，可根据游戏设计方案或视屏需求条件生成游戏相机的新实例。另外，CM 还将处理切换操作、插值计算、优先权控制（通过提示管理器，具体内容稍后解释）、控制器输入管理、回放模式以及时间控制。

需要说明的是，CM 中定义了一个 Update()函数并可从游戏的主循环结果中加以调用，该函数将使用到距上次更新调用的时间流逝值并将其作为参数加以传递；通常，CM 还将使用到基于一个或多个玩家的控制器输入数据。deltaTime 值随后将被传递至相机的 Update()函数中。

针对相机管理器中的 Update()函数，下列内容对其主要步骤进行了简单的总结：

- ☑ 根据脚本需求条件开启/移除游戏相机。
- ☑ 更新活动相机（由于插值相机常依赖于其他因素，因而往往得以最后执行）。
- ☑ 将控制器输入内容传递至活动相机中。
- ☑ 更新音频系统的位置和速度数据（即多普勒效应）。
- ☑ 针对当前视口构建渲染环境。

下面将对各步骤进行逐一考察。

相机更新循环结构

相机管理器中的主 Update()函数可分解为多个独立步骤。其中，相关步骤的执行顺序常保持固定不变，尽管某些游戏并未遵循此项规则。

- ☑ 更新活动相机。针对于处于激活状态的内部相机表，其中的全部相机均调用各自的 Update()函数，通常情况下，该函数将采用流逝时间值作为传递参数。另外，相机逻辑的执行顺序可显式地加以定义，抑或仅采用相机激活时的顺序。通常，考虑到插值相机通常取决于源、目标相机更新后的位置和方向，因而此类相机将最后执行其逻辑内容。另外，在某一时刻，相机的激活数量则不做任何限制；处于激活状态下的相机通常并不知晓是否已绑定至某一视口之上。
- ☑ 更新相机脚本。此处，可根据游戏体验需求条件对相机实施添加或移除操作。
- ☑ 输入处理机制。该机制负责处理调试型相机的功能以及第三人称相机的控制行为。
- ☑ 预判逻辑。该功能多用于启动电影视频序列，相关序列一般出现于更新循环的开始阶段。
- ☑ 判断顺序。某些相机通常会依赖于其他相机的行为方式，如插值相机。针对于此，CM 可确保相机获取正确的更新顺序。
- ☑ 电影相机的延迟效果。同样，电影相机亦应在更新循环的开始阶段启动，并终止于游戏帧的结束阶段，此时，其他相关事项均已更新完毕。
- ☑ 后置判断逻辑（外部依赖性）。针对于正常游戏更新循环中的 CM，某些系统则在其后运行。针对于此，可后置判断逻辑，进而收集外部依赖关系所生成的数据信息。
- ☑ 更新音频逻辑。根据空间状态，可执行多普勒效应等操作。
- ☑ 调试型相机。调试型相机的处理方式则稍显不同，考虑到相关代码无须出现于成品游戏中，因而可对其执行条件编译。
- ☑ 创建渲染器。该过程常会涉及相机的抖动操作。

提示管理器

各 CM 均与相机提示管理器关联（CHM），且该管理器继承于通用游戏提示管理器，并可视为脚本系统与 CM 之间的接口。同时，脚本系统亦提供了相关机制并提供提示管理器控制、调整相机的行为。提示管理器可实现下列操作，即定义当前活动提示对象的优先权、开启/禁用插值计算、调整活动相机等。

不难发现，该类继承于更为通用的解决方案，且相关问题常会包含基于各类游戏系统的覆写操作，即 CGameHintManager 类。基础类 CGameHintManager 针对于提示对象的激活/禁用功能提供了对应的框架、属性以及其他混合管理功能。同时，CCameraHint 类也继承自基础类 CGameHint，且可用于玩家提示对象。

关于脚本方案以及相机脚本特定方案，读者可参考第 6 章以获取更多内容。

震动管理器

当模拟玩家角色的震动效果以实现脚本或实时游戏事件的反馈时，震动管理器提供了与此对应的相关机制。同时，震动管理器还将负责检测处于激活状态下的震动器；在一段时间内更新振幅并相应地调整内部震动机制。这里，视口管理器将查询震动管理器并在适当的时候将其作用于当前活动相机之上（但不包括调试型相机）。

相机震动操作

相机的震动效果突出在渲染阶段执行，而非在游戏场景中真实地移动相机。这将导致场景视野与几何体相交，严重时甚至可错误地显示游戏场景的外部区域，该问题可根据当前空间范围对相机的振动量按比例进行调整。通常情况下，震动转换可在相机的局部空间内计算，随后执行渲染操作。需要注意的是，该计算过程并不会影响到 HUD 以及其他 2D 视觉元素的渲染操作。另外，尽管震动多用于表现武器设备的反作用力，但震动的方向通常沿屏幕的水平或垂直空间轴进行。

对应于相机的各个空间轴，其震动位移通常包含 3 个分量，沿各轴向的振幅也可通过多种方式加以控制。

- ☑ 正弦波。振幅以及运动周期也可在一段时期内发生变化。
- ☑ 振幅映射。这里映射函数（如分段式 Hermite 曲线）可将时间数据映射至相机各局部轴向上的位移值。
- ☑ 随机噪声。定义某一位置并用于搜索或生成基于样条曲线上的控制点。

相机震动效果往往用于增强游戏场景中的真实感受，如爆炸效果、庞大物体的移动效果（如 Boss、机械装备）、玩家角色被击中时的效果、武器设备的反作用力、人物跌倒/跳跃时的着陆效果等。因此，此类功能多出现于第三人称相机中，但需要对其加以谨慎处理。

经转换后的相机在其渲染过程中可能会过于接近几何体（静态或动态），进而会与相机的近剪裁面相交，抑或相机的视锥体延展至场景环境的外部空间，从而导致渲染画面出现问题。

该问题可通过光线投射机制或（在第一人称相机中）将震动量限制在玩家碰撞体的尺寸之上。

隆声/力反馈效果控制器

某些情况下，隆声效果可与相机的震动效果或其他游戏事件结合使用，需要注意的是，并非所有控制器均支持此类力反馈或其他类型的触觉技术。相反，某些基于游戏机的、低成本的解决方案则采用偏移加权马达方案，其旋转速度通常处于可控状态。通过调整控制器中的马达旋转速度，可实现彼此各异的隆声效果。

然而，马达通常具有较长的加速旋转或减速旋转时间，因而难以描述操作过程中的细微变化。实际上，较为常见的做法是仅开启/禁用马达，并通过切换频率定义玩家体验过程中的隆声效果。从理论上讲，上述反馈效果可视为一类多通道混频器，且与混音机制具有异曲同工的效果。这里，隆声量可视为一类 ADSR 包络（时间与隆声振幅）。具有真实力反馈功能的控制器通常会提供预置效果的可选方

案，并可对其实施编辑操作。

从技术角度上讲，隆声机制并非是相机系统的必需方案，但该方案依然可在震动管理器中进行初始化以及更新操作。实际上，隆声效果的振幅常可根据震动管理器中的振动值（之和）加以计算。

游戏相机

游戏相机可视为相机系统中的基础模块，其实现方案亦多种多样，继承机制以及组件机制可视为两类主要方案。

CGameCamera 类包含了丰富的信息，并可实现相机的定位以及对象操作，但该类通常不包含除视域之外的其他渲染数据。其中，视域可通过水平或垂直方式加以定义，并与视口的宽高比结合使用。此处，视口用于显示相机视觉画面并定义其他轴角（读者可参考第 10 章以了解对应的计算方案）。通常情况下，可采用垂直方式定义视域，与传统的 4:3 电视标准相比，该方案则更适用于宽屏电视，进而显示相对宽广的场景画面。这里，相机类可视为一类虚拟对象且可通过多种方式生成游戏场景的视觉画面。然而，相机无须了解数据与渲染操作之间的解释方式。

一旦定义了基础相机类型，即可通过加载移动和逻辑函数从该类中继承特定的相机行为。除此之外，还可定义相应的接口类 IGameCamera 进而实现全部所需功能项。考虑到相机的行为多通过其移动特征加以定义（包括其定位方案），因而可用于区分不同的相机类型，例如 CpathCamera、CSpindleCamera 等。若上述类均包含相应的标准访问方法以获取有效的移动和方向数据信息，则可于随后实现对应的插值类，且无须考虑插值相机的具体类型。

继承的相机行为

一类较为常见的类实现方案是采用基于相机类的层次结构，且后续各类均具备各自的行为特征。以下内容显示了典型方案的相关结构：

- ☑ 角色。
- ☑ 相机。
- ☑ 第三人称相机。
  - ✧ 伺服相机。
  - ✧ 观察相机。
  - ✧ 路径。
  - ✧ 对象表面。
  - ✧ 静态相机。
  - ✧ 切换操作。
  - ✧ 插值计算。
  - ✧ 调试机制。
- ☑ 第一人称相机。

在此类实现方案中，各个相机属性期望集可整合至某一独立相机继承类中，且与特定行为相对应。

针对于上述继承类，较为常见的方法是使用基类 CGameCamera，鉴于相机行为的特殊性，因而无须生成派生子类。

基于组件的相机行为

基于组件的相机行为可视为类继承方案的替代方案，实际上，游戏相机的各类属性可分解为多个彼此交互的组件，且每一组件称作“行为”。针对于此，上述行为可组合使用，甚至可在运行期内发生变化，进而生成不同版本的相机对象。而且，与层次结构相比，此类设计方案更具数据驱动特征，其中，特定行为可通过统一接口加以调整，进而使得相机的扩展操作不会对现有相机功能产生任何影响。另外，基于组件的相机行为的另一个优点是保持相机使用一致性，进而可在游戏体验过程中简化相机的脚本应用。

常见的相机行为包含以下内容：

- ☑ 相机期望位置的确定方案。
- ☑ 基于期望位置的移动方案，包括碰撞检测以及反馈机制。
- ☑ 对象方案。
- ☑ 相对于期望方向的旋转操作。
- ☑ 插值计算（朝向或背离当前相机）。
- ☑ 视域（并结合取景操作）。
- ☑ 针对特定相机视见内容的渲染效果。

其他相机属性也可一并视为组件，但有时无法证明它们是一类完全独立的行为；相反，可将其整合至某单一行为中，如位置约束条件。这一类约束条件通常具有一定的独立特征，其应用方式往往与其行为无关。然而，考虑到某些约束条件无法应用于特定的行为环境，因此，特定行为的初始化过程将会禁用某些不适宜的约束条件。

当采用基于组件的应用方案时，虽然单一组件可设计为独立型组件，但某些时候，特定组件（如方向行为）常与其他组件的状态相关（此处为位置数据）。因此，针对组件的更新操作，存在一类隐式顺序以实现该操作。而且，组件往往需要相关方法以对其他组件实施检测。此处，无须使用局部存储指针，可简单地向行为函数传递一个指向父相机的引用并对该行为进行检测。

电影视频相机

尽管多数时候可将视频相机与其他相机同等对待，但对其加以独立考察将更具优势。首先，游戏相机与视频相机之间可实现持续的并行操作，并支持两种类型相机之间的无缝转换。其次，游戏逻辑多与其游戏相机相关，且可持续发挥其功效并与渲染游戏机场景的视频相机无关。最后，视频相机可用作常规游戏相机所渲染的场景元素，例如，可将场景元素渲染至纹理之上并用作游戏内部的部分场景效果。此类分离方案也可扩展至视频相机的数据访问方式，为此，可针对当前相机（如在相机管理器中）设置相关访问函数，进而显式地控制视频相机。在使用视口的游戏渲染体系架构中，将视频相机设置为独立的视口通常十分有效，通过该方式，可适当地调整视频画面内容且不会对常规游戏体验产生任何影响。

调试型相机

通常情况下，出于调制目的，可在相机管理器中单独设立相机，这一类相机常称作调试型相机，该类相机继承自基础相机类且仅用于渲染操作，本章后续内容将对其具体应用进行详细的介绍。

脚本机制实现方案

第 2 章曾指出，根据游戏环境的变化或出于强调游戏体验的特定元素的目的，有必要针对相机行为实施动态调整，该机制可简单地描述为脚本机制，尽管该机制无法对某些预定义方案所产生的变化进行较好的限定。这里，改变相机行为的脚本对象常称作“相机提示”对象，亦即，根据处理特定环境或游戏体验需求条件的方式，这一类对象常会向相机系统提供辅助信息（即所谓的“提示线索”）。另外，对应于相机的行为类型，相机提示对象通常包含不同的属性，因此，可向设计者提供相应的相机提示对象调色板。

同时，常见的相机提示对象实现方案则是使用多态并从基础游戏提示类中继承各个特定的提示对象；另外，底层类负责处理基于常用对象类型的特定操作，相关类均继承自上述类，例如：

- ☑ 不包含位置信息的对象（实体）。
- ☑ 包含位置和方向信息的静态对象（角色）。
- ☑ 受外力支配的移动对象（物理角色）。

这里，有必要对对象进行适当分类，进而减少特定对象的定位时间；因此，对象可通过特征表加以引用（例如 AI 对象、包含碰撞信息的对象等）。脚本对象应提供对象的添加/移除功能、基于唯一标识符（对象创建时进行赋值）的对象搜索功能、依据优先权的对象逻辑排序功能等。除此之外，相机管理器还用于维护潜在相机提示对象表，以及处于激活状态下的对象表。

相机脚本对象

游戏中往往包含了各类脚本对象，且该类对象多与游戏风格相关。然而，某些脚本对象类型则较为常见（与游戏风格无关）且与相机系统紧密相关。下面将考察此类对象的实现方案。

相机提示对象

相机提示对象针对相机属性加以描述，并用于显示游戏环境或游戏体验元素中的特定内容。相关对象一旦处于激活状态，即向提示管理器告知其状态，进而控制对象的优先权以及游戏相机对象生成操作。相机提示对象可视为简单的数据重定位操作，且与活动状态的管理过程不同，该操作并不涉及过多的真实逻辑。相对于相机系统而言，提示对象的使用目的则稍有不同，如调整玩家的控制操作、禁用玩家的某种功能等。这里，建议实现基类 CGameHint 并派生相应的 CCameraHint 和 CPlayerHint 对象。若此类对象仅保存数据，则只需持有唯一的数据标识句柄即可，进而可从通用游戏对象数据结

构中获取相关数据。

现代游戏引擎通常会在游戏过程中动态地加载或卸载数据资源，其中包括相机提示这一类数据对象。因此，确保相机提示对象的引用操作可有效地禁用相关对象。针对于该问题，一类处理方案是使用脚本对象的消息处理机制，进而使得对应对象接收结束消息。随后，脚本对象将通知相机提示管理器相关提示对象处于无效状态；除此之外，提示管理器还可在每帧中激活内部对象表中的相机提示对象。该机制在非产品环境中十分有效，实际上，它也存在于游戏的最终版本中。某些时候，提示脚本机制的生存周期往往大于脚本对象自身的生存周期。此时，应在提示管理器中保存提示数据的局部缓存副本。

触发体

触发体对象可视为一类三维空间体并用于检测对象自身的入口/出口状态。同时，游戏消息被发往至其他游戏对象，如相机提示对象。这里，触发体通常实现为简单的几何体形状，如轴对齐包围盒、圆柱体或球体。某些情况下，有必要从多个触发体中（通常处于叠加状态）构建合成对象。另外，还可定义子空间体并用于确定外部大型空间体的局部区域。此时，子空间体可在大型空间体内部实现一类“钻孔”效果。因此，触发体与结构实体几何体（CSG）技术具有某些共性，其中，Boolean 操作符（如并集操作、交集操作以及差操作）可将简单的几何图元整合为复杂的对象。

渲染/光照效果

游戏场景的特殊变化常通过游戏脚本机制加以定义。类似地，游戏场景的光照机制也可通过脚本加以实现（环境光以及特定光源）。某些渲染效果也可通过以下方式得以完成，即向相机发送脚本消息、相机检测游戏状态，或激活采用特殊渲染效果的相机对象。常见的基于相机的渲染效果包括热力成像、X 射线模式以及夜视效果。

消息延迟

延迟可视为一类简单的游戏消息整合机制以简化脚本机制所产生的变化，延迟机制可向单一脚本对象提供相应的功能项并反馈于某一消息集，进而将消息传递至其他对象组中。当然，上述过程可通过更为直接的关系方式加以实现，消息传递条件的改变将使得多个对象发生变化。在消息延迟机制中，仅单一脚本对象的连接关系需要更新（亦即延迟对象）而非全部与该变化相关的对象（即与延迟对象存在连接关系的对象）。

序列事件计时器

通常情况下，可定义多个游戏事件且此类事件通常以特定的时间间隔或顺序出现，序列事件计时器可实现消息的时间排序，或使其在同一时刻以特定顺序出现。因此，脚本对象应允许事件标记根据时间轴设置相关时间，且相关标记与特定的消息关联。

通用路径

多数游戏均会使用到游戏场景路径，进而控制对象的移动和方向，这一类路径可通过多种方式加以定义，具体内容可参考第 8 章。这里，为了避免多个游戏对象复制同一行为，一类更加高效的方法则是令单一对象定义路径，并可针对其他游戏对象计算位置数据，其计算过程通常采用两种方式，即定义时间值（基于路径遍历的全部时间值）以及定义距离值，此处，两种方法均需要对开始点和结束点进行预定义。针对封闭循环路径，需使用连接信息以定义路径的计算方向。

封闭循环路径将使得问题变得复杂化，其中，循环顶点可能为连接开始位置和结束位置（形状近似为数字 6 的路径），除非连接点自身被视为路径的结束顶点。另外，循环位置也可作为独立的控制点或仅作为一个标志加以处理。针对于前者，内部逻辑则稍显简单，但却会改变循环位置处的路径曲率；而后一方案则会在计算路径时涉及更多的内部管理机制，但若将其抽象至某一独立函数中，并从任意既定索引值中获取控制点，则上述问题可以得到很好的解决。

脚本逻辑排序

类似于其他游戏对象逻辑，脚本逻辑通常也在游戏更新循环中的同一位置加以执行，若脚本逻辑致使消息被发送，则此类消息将即刻发往对应的接收器中。需要注意的是，接收消息的对象可能在当前游戏更新操作中执行了其自身逻辑，因而会在下一循环操作过程中对状态变化有所反馈。同时，消息处理代码将对状态变化做出适当的调整。除此之外，还可对消息进行缓存，当主对象逻辑执行完毕后即对其实施发送操作。第 2 章曾讨论了常见的游戏更新循环操作。

消息机制

针对于某一游戏对象，通常有必要向其他游戏对象告知事件的发生事态，该情形可通过消息系统加以实现，其中，对象可在当前条件下向其他对象发送消息（即与事件相关的数据）。自然状态下，该过程需在场景编辑器中确定相关方法以显式地定义该行为，抑或隐式出现于游戏体验过程中。针对于后者，当满足相关条件时，场景编辑器依然需要对消息类型加以定义；考虑到接收对象可能根本不存在于场景编辑器中（亦即，该对象在游戏中产生），因而无法显式地定义两个对象间的相互关系。需要说明的是，根据触发消息的事件、发送消息的对象以及导致该消息发送的潜在不同对象，消息通常包含了与此相关的诸多信息，如进入触发体的玩家对象，将导致触发体脚本对象向与其关联的任意游戏对象发送“进入”消息（相关对象在场景编辑器中加以定义）。这里，消息发送者为触发体对象，而事件的始发者则是玩家对象。此类事件将导致生成敌方角色、激活某一谜题、播放视屏序列等。

优先权

考虑到相机提示对象的重要程度，各个相机提示对象均会设置某一整数值与此对应。在该方式中，可根据游戏体验需求条件激活并操控多个游戏提示对象，例如，在特定的游戏区域中，相机可靠近玩

家角色，然而，其部分区域中很可能还包含进一步的限制条件。

另外，优先权还可根据相关属性加以设定，而非简单的数值数据。例如，两个相机提示对象可被同一游戏事件同步触发，相机提示对象的选取方案可根据玩家与提示对象或其他游戏对象的空间位置关系加以确定；又如，取决于玩家与触发体之间的相对移动方向，包含一致参考帧的相机提示对象将被优先选用。

游戏中可能存在大量的区域需要架设“专职”相机且相机间彼此邻接或相互交叠，因此，应采取相应的优先权方案以确保获取正确的相机行为（而非简单的覆写操作）。通常，该过程通过相机管理器（或相机提示管理器）加以处理，并涉及各个相机提示对象。

针对于此，可简单地使用排序数组（如 C++语言中的 STL 机制）并将活动相机置于活动相机表中的首部位置（或尾部位置）。同时，可采用比较函数定义具体的排序操作，该函数可根据潜在的提示类型进行覆写（亦即，与玩家提示相比，相机提示往往包含不同种类的比较函数）。

插值计算

脚本相机之间的插值计算多出现于相机提示对象或其他脚本事件中，因此需要对插值方式进行相关定义。插值过程中的可控相机属性包括位置、方向以及视域等内容，且线性插值并非必需。相反，映射函数可采用插值时间作为输入数据，并根据相关参数曲线返回插值后的结果值（位置、角度等）。另外，插值相机可定义为引用了两个其他相机对象的脚本对象，进而对插值过程实施有效的管理。关于插值计算，读者可参考第 8 章以获取更多内容。

性能问题

相机逻辑通常会产生较大的计算开销，当考察游戏机场景环境并针对潜在相机位置确定碰撞或路径数据时尤是如此。考虑到有限的运行期处理器资源，游戏间相机系统的运算量也彼此不同。处理器分配问题一直被程序员视为重要课题，且此类问题短期内并不会产生太多变化，相关变化也仅是对新功能稍作适当调整而已。针对于处理器需求，活动相机近似等同于复杂的 AI 角色对象。

分摊机制

复杂的游戏系统常会根据相机逻辑采取相应的分摊机制，并对中间结果实施缓存以备后续更新操作（或插值计算）使用。考虑到相机的反馈频率相对较低（与计算机计算速度相比），因此可在连续的更新操作中采用光线投射或其他蕴含较大开销的计算操作。与玩家输入过程不同，相机变化可出现于多次更新循环中，且与相机移动相比，漫游信息可以较低的速率进行更新，当此类信息保持相对固定时尤是如此（如场景几何体之间的光线投射计算）。通常情况下，考虑到相机整体的平滑移动，其移动行为通常不采用分摊机制。需要注意的是，分摊机制有时会对相机与游戏变化之间的反馈行为产生较为明显的影响，因此，依赖于其他对象的属性数据通常无法正确地反映分摊机制所产生的计算结果（如根据玩家角色位置的距离计算）。

预处理机制

环境可有效地减少相机逻辑在运行期内的 CPU 计算开销，针对于相机而言，CPU 性能往往体现在针对游戏机场景或游戏对象的光线投射或碰撞计算之上。针对于此，相机沿既定路径的移动行为可有效地减少或消除上述问题所产生的负面影响。

开发工具

多数相机逻辑多依据正常的游戏体验需求加以编写，因此，基于相机的游戏开发工具屡见不鲜。多数相机行为的变化均与第三人称相机相关，如前所述，偶尔情况下，第一人称亦会产生行为变化。无论如何，可通过场景编辑器定义这一类变化，亦即，可通过游戏外部应用程序对游戏元素进行整合、调整。需要说明的是，现有建模应用程序多支持上述功能（如 Maya），但多数公司均会开发适合于自身风格的建模工具，进而获取期望中的视觉效果。

下面将对开发工具与相机行为之间的定义方式进行简要的分析。

场景编辑器

相机编辑工作多出现于场景编辑器中，至少针对交互式序列而言情况大抵如此，当然，基于游戏要求驱动的非交互式序列也可通过此方式加以定义，抑或使用单独的建模软件包。场景编辑器将针对游戏对象定义唯一的相机属性，并进一步确定对象间的相互关系，例如，受限于某一路径上的相机应具备定义路径或链接至其他外部适宜对象的能力。

若脚本系统可通过场景编辑器设置或控制游戏场景中的相机对象，则该工具应可提供一定程度上的相机行为的预览功能。这里，构建正确相机行为的循环迭代过程可视为相机开发过程中一项较为耗时的任务，当采用游戏引擎渲染视频序列时尤是如此。此时，提供某种形式的、基于编辑器的相机操控将变得十分重要，该操作可以在目标平台上执行，或在编辑器内部进行渲染。

当相机与路径结合使用时，相关路径应在编辑器中可见（使用与游戏自身相同的计算逻辑）并易于编辑——理想状态下，此类操作应可在运行期内执行。鉴于编辑器内的游戏逻辑预览操作并非总能实现，因而需要采用其他方法模拟对相机移动所产生的影响。从视频序列中可清楚地看出，可设置计时器并对序列内的事件加以控制。

除此之外，当相机针对玩家位置（如场景内的 3D 样条曲线）实现反馈时，应对此辅以其他计算方案。编辑器内游戏数据的直接更新操作（当目标平台处于运行状态时）对于整体任务而言十分重要。

总而言之，当定义或编辑相机行为时，场景编辑系统可视为一类重要的开发工具，其重要性主要体现在：对于频繁执行的相关任务而言，场景编辑器可提供方便、快捷的操作界面，相关示例包括相机路径的定义（路点的设置以及连接关系）、相机提示对象的设定以及属性的编辑、脚本对象的预定义宏等。

针对于相机系统，下列内容列举了场景编辑器可提供的期望特征：

- ☑ 相机的定位以及定向。该操作可视为一类较为基础的功能项，需要说明的是，相机的静态定位方式仅是诸多相机需求条件的一个子集。
- ☑ 相机属性编辑。针对于其他游戏对象，编辑器应支持源于设计人员的、个体相机属性的修改操作。理想状态下，此类变化应立竿见影并在游戏运行时加以更新（尽管实现起来较为困难）。
- ☑ 从相机角度进行观察。如果相机的渲染画面通过编辑器主窗口或弹出窗口向当前设计人员加以展示，则在编辑器内调整相机方向将变得十分重要。需要注意的是，宽高比以及显示内容的视域应与真实相机匹配。
- ☑ 在游戏内部实施编辑工作并于随后对数据加以转换。尽管该操作并非常见特征，但却可对目标平台上或游戏内的某些相机属性实施有效的编辑，并于随后将修改后的信息发回至主机或数据文件中（针对整合后的脚本或代码）。
- ☑ 可限制某些属性以显示期望行为。仅在既定时刻显示相关属性。
- ☑ 路径——自动路点丢弃以及连接功能。针对路点丢弃操作，编辑工具的自动操作功能将会十分有用。例如，当玩家遍历某一区域时，可丢弃相关路点并对此类信息进行收集，以备后期路径转换时使用。
- ☑ 空间体。针对于各类相机相关功能，空间体提供了定义 3D 区域的能力。
- ☑ 表面。此处，该数据形式可用于定义相机的移动表面。
- ☑ 连接至目标对象。该功能可针对特定相机定义对应的目标对象。
- ☑ 一段时期内的位置/方向/倾斜/视域控制操作（样条编辑器）。大多数开发工具内均设有样条编辑器，进而可直接调整相关数据值，其中包括位置、方向、倾斜、视域等内容。在某些环境下，手工调节样条曲线则十分有效。
- ☑ 一段时间内的目标对象估算以及插值计算。当对象与相机行为发生关系时，该操作反映了一段时间内该对象的最终位置。

相机碰撞网格

类似于其他游戏系统，相机系统也将采用相同的场景环境碰撞数据；另外，若相机包含自己的碰撞几何体，则碰撞计算的实施过程将变得更加方便、合理。如前所述，与常规碰撞几何体相比，该类几何体通常更加简单，且场景编辑器常会提供相应的工具以使设计人员可对其进行调整，此时，外观上的变化并不会影响相机的碰撞约束条件，尽管该操作应谨慎处理以确保相机几何体处于更新状态。

相机调试技术

相机系统通常较为复杂且需要进行反复调试，在开发的初始阶段尤是如此，一旦相机系统构建完毕，则应辅助设计人员实现其期望中的相机方案。另外，应确保用户自定义相机能以扩展方式进行工作。进一步讲，调试功能还可有效地改善性能问题，这里，存在多种方式可实现上述目标，且多数方案取决于相机系统所采用的规范及其所处的具体环境。同时，相机的调试过程还可简单地视为一类交互操作并包含诸多优点，例如，可方便地捕捉相机系统的信息并在运行环境外部对其进行检测。除此

之外，一类混合方案则对一段时间内的用户输入数据进行捕捉，并对部分游戏内容加以回放。

下列内容则列举了游戏产品中相对成熟的调试技术，尽管未曾提供具体的实现细节（相关细节往往依赖于游戏引擎以及对应的开发工具），但相关信息足以表明特定环境下所涉及的功能项。

交互式调试技术

一类较为方便的调试方式是在相机系统运行时检测其状态，该方案并非仅通过调试器应用程序察看其内部状态，通常情况下，还可将其他相机数据表现元素添加至游戏场景渲染画面中或输出为纹理数据。这一类文本输出可发送至主显示设备或其他外部目标设备之上（如网络 PC）。

下列内容显示了应用程序运行时的备选调试技术方案。

内部属性查询

如前所述，该操作可对相机系统进行直接察看，并可设置断点以暂停当前游戏流程，然而，该过程依然会产生某些问题。首先，执行代码应包含调试信息，这大大地增加了代码的尺寸；其次，执行代码与调试器之间的通信常会产生额外的性能开销。上述两种问题均可对相机系统的行为产生潜在的影响，但针对大部分相机问题，上述方案通常十分有效。下列内容列举了一些需实施有效跟踪的相关属性。

- ☑ 按优先权排序的活动相机。
- ☑ 当前相机，包括其类型和行为。
- ☑ 目标对象（若存在）。

独立型调试相机

针对于相机系统（实际上则是通用相机调试机制），常见的调试辅助方案则是维护并更新独立的调试型相机，且该相机通常位于游戏的开发版本中；同时，该相机仅用于渲染目的且不会对其他游戏对象产生任何影响。一般情况下，用户可对相机的位置和方向加以操控，且与主玩家角色或其他游戏对象的控制操作无关，具体操作过程可按如下方式加以实现，即采用独立输入设备，或将游戏模式设置为将控制器输入直接对应于调试相机，而非当前游戏。针对于后者，应确保调试型相机获得的输入内容未被常规游戏体验中的控制变化过滤掉。总而言之，调试型相机的更新操作不应受到游戏状态变化的任何影响。当调试型相机与游戏执行速度的调节功能结合使用时，其有效性不言而喻；同时，用户还可在常规游戏相机与调试型相机之间实施快速切换，当采用同一控制器时尤是如此。

这里所阐述的多数交互式技术其最佳视角均为调试型相机所提供的外部视角，需要注意的是，调试型相机不应对视见剔除操作以及其他渲染优化操作产生任何影响（亦即，仅向渲染系统提供位置和方向数据），针对于确定游戏场景的渲染内容，此方式提供了一类颇为有效的操作机制。

当使用调试型相机时，可添加某些特定功能项以辅助观察者实现有效的操作，相关内容包括：

- ☑ 将玩家角色重定位至当前相机位置处。
- ☑ 显示当前相机位置（该操作对于 QA 处理十分有效）。
- ☑ 沿调试型相机的前视向量投射光线，进而确定环境属性（如表面碰撞信息、材质属性等）。

☑ 操控调试型相机时，应暂停当前游戏处理过程。

☑ 获取当前渲染缓冲区中的内容并将其导出至外部存储中，进而实现截屏功能。

调试型相机多出现于处于活动状态下的游戏环境中，因此，此类相机通常与常规游戏显示共用同一视口，尽管为调试相机设置独立视口并非难事。当在多显示（或多窗口）环境下进行观察时，多视口方式则显得异常有效，当游戏处于运行状态时，其中一人可作为玩家对游戏状态进行察看，而另一人则可考察某些特定的游戏体验元素。

控制游戏刷新率

调整游戏的刷新率可视为一类较为常见的调试技术，针对单步更新循环尤是如此。标准调试器通常包含此项功能且易于实现，但该功能主要面向非专业人员或者缺少调试器时。这里，降低游戏的刷新率可通过以下方式实现，即在后续更新操作中添加时间延迟，但仍向游戏对象传递原（固定）时间步。除此之外，还可减少时间步自身长度值，但这将导致真实的游戏体验过程产生一些误差。

通用相机状态

当游戏处于活动状态时，可定义多个状态变量以对其行为实施有效的跟踪，某些变量则与脚本方案相关并用于确定活动相机，且多出现于相机管理器中，而非某一特定相机，下列内容列举了对应的可跟踪项：

☑ 状态信息（例如，是否处于活动状态、是否处于玩家操控指向等）。

☑ 脚本消息机制（接收后的消息或发送后的消息）。

☑ 相机提示/游戏相机的变化。

☑ 遮挡状态。

☑ 检测条件的活动状态；当发生某一事件时，是否需要相机实施重定位操作（同时显示相关事件的特征）。

☑ 无效的相机属性，包括转换矩阵的有效性。

相机属性的视觉表现

这里，观察相机属性在一段时间内的变化方式通常大有裨益，在交互式环境中，该操作则可通过绘制场景中属性的视觉表现元素加以实现，相关属性包含以下内容：

☑ 处于活动状态下的相机的位置数据（可采用线框球体或立方体加以显示，用位置之间的链接显示移动方向）。

☑ 当前相机方向（例如，源于当前相机位置并通过彩色箭头显示相机居中的前视向量、Up 向量以及 Right 向量）。

☑ 期望视点位置。

☑ 期望方向（不包含特定的视见位置）。

相关属性的显示功能通常十分有效，例如，根据特定分类或相机对象，可实现属性显示的动态过

滤机制。如前所述，还可针对相机属性打印其文本信息。

属性滞后现象

类似地，察看相机属性的最近历史记录也是一项十分有用的功能。通常情况下，可在每次更新操作中捕获全部期望属性数据，并在游戏执行过程中显示其子集内容，然而，内存或性能问题往往限制了显示属性的数量，针对于此，存储数据以备后用不失为一种处理方案。当采用此类方式进行察看时，下列内容显示了一些较为常见的属性：

- ☑ 相机位置。由于相关位置数据多以均匀间隔量加以分布，因而它们有时亦称作“面包屑”数据（亦即，仅当相机距前次存储位置之间的距离超出阈值时，方对其加以记录）。若经特定位置后的时间量较为重要，则上述“面包屑”数据将会发生变化（例如，改变渲染表达结果的比例或颜色）。实际上，针对于相机设计人员而言，“面包屑”数据的改变提供了一种快速的属性评估方式，抑或游戏场景内相机位置的反馈方式（如视线问题）。另外，显示存储位置之间的连接信息亦十分有效，相关连接结果多呈现为直线状态；同时，位置间隔还体现了相机的瞬时移动行为。
- ☑ 相机方向。考虑到方向变化的本质特征，其一段时期内的变化方式通常难以辨别。当与“面包屑”位置数据结合使用时，则可得到一类方向变化的近似解决方案。其他表现形式还包括，通过单位球表面上的顶点显示方向变化。另外，方向变化还可通过欧拉角加以跟踪，并作为直方图加以显示（亦即，分别采用时间轴和角度值加以表示）；同时，通过类似的方式，建模软件包使得美工师可对一段时期内的旋转加以定义。当尝试跟踪插值问题时，此类方案将十分有效。

移动约束条件

考虑到相机位置往往受限于其行为方式，因此应对相关限制条件加以考察，尤其是某些“软”约束条件，具体示例稍后解释。

- ☑ 此处，可根据样条曲线以及线段的近似表现方式绘制移动路径，其中，线段数量的选取方案应满足美学以及性能需求，且不应对最终的游戏投影产生任何负面影响。
- ☑ 同时，移动表面则可通过线框方式进行绘制，进而避免对其他游戏元素产生遮挡现象，以便更好地理解相关表面的形状（尤其是曲线）。

视线

相机与目标对象间良好的视线关系将有助于实现正确的跟踪计算，这里，可沿相机的前视向量绘制一条直线并以此显示视线。另外，为了进一步展示直线的状态，可使用不同的颜色对其进行标识，例如，当视线受阻时使用红色，否则使用绿色加以显示。同时，另一相关属性“角色遮挡”也可通过类似方式加以显示。这里，若直线绘制长度有限，则还应显示对应的断点（碰撞几何体通常不可见且与渲染几何体截然不同）。其他有用信息还包括对象的材质属性、几何体与投射光线之间的阻挡关系（如

岩石、相机碰撞、天花板、地板等)、对象实例的名称等。相对于观察者而言，为了有效地防止信息过量，可设置相应的调试选项，进而可对信息量做适当调整。

基于行为的渲染操作

某些相机以及行为（如第 7 章所描述的纺锤体相机）通常会包含特定信息，并用于确定其期望位置、方向等，相关行为的计算方式应予以说明。在纺锤体相机中，潜在移动范围常与相机半径插值计算结合使用，因此，基于属性确定方案的相机行为常会使用到相对独特的渲染技术，例如，流场和排斥体即需要较为特殊的处理方案，进而展示与当前之间的作用关系。

脚本调试

第 6 章曾对此有所提及，下面将针对该话题进行简要回顾。

- ☑ 脚本语句的执行/过滤机制。针对任意游戏脚本（或当前活动脚本对象的其他子集，且由期望过滤机制加以定义），应可显示各执行语句的历史记录，进而替代调试器。
- ☑ 调试信息日志。脚本对象应可生成相应的调试文本信息，以显示其内部运行状态。
- ☑ 消息过滤机制。如上所述，应可通过调试消息日志实现对象间的通信以及显示过程，除此之外，消息类型的过滤机制亦不可或缺。
- ☑ 对象状态。个体对象类型常根据其相关行为报告相应的内部状态信息。

数据日志

在游戏开发过程中，数据日志可视为一类常见的技术，其中，属性值在一段时间内被记录并输出至游戏的外部存储中（通常无此必要）。此处，捕捉后的数据多与特定的时间值或输出记录相关，进而可确定事件的排序。一旦获取此类数据，即可针对特定信息对其加以分析，并传递至其他应用程序中（如制表软件或可视工具）。

当然，日志也可通过较为特殊的方式加以执行，如采用简单的 printf()文本输出函数。即使如此，精简后的信息量依然可保证排序事件的有效分析过程，其粒度远小于单步更新循环操作。另外，当调试操作无法得到保证时，日志机制将起到很好的辅助作用（如 QA 处理过程）。需要注意的是，将尽量减少日志对游戏性能所产生的影响（以避免出现所谓的“观察者”效果，其中，观察行为将修改潜在的输出结果）。针对于此，一类优化方案则是本地缓存日志信息（假设存在相应的内存空间），并在适当时刻且不影响游戏性能的前提下将日志信息发送至外部存储中。

一旦获取了这一类数据，则可通过多种方式对其实施可视化操作。因此，文本数据的察看操作可视为一类最为简单的方案，且在多数时候可满足相关需求。即使缺少可视化工具的支持，也可方便地将数据导入至制表软件或现有应用程序中，进而实现日志数据的图形化或类似表现风格。当数据相对简单时，此类方案则较为适宜（如跟踪一段时间内浮点数的变化）。

然而，可视化工具依然具有较高的实用价值，当对较大的采样集加以分析并显示一段时间内的变化状态时尤是如此。例如，考察相对简单的游戏对象的位置和方向信息，此类信息可导入至场景编辑

工具中，进而可方便地观察一段时间内对象的移动状态。另外，针对于游戏环境中的显示对象，这一类显示方案还有助于观察者理解对象的错误行为方式。不难发现，日志方案广泛地应用于通用游戏调试机制中，且要求设计人员具备相应的程序设计能力。针对于此，存在一类中间件解决方案进而可降低具体实现过程中所产生的计算开销。

游戏回放机制

交互式或日志技术的一类替代方案是游戏回放机制，用户可通过该方式对相关变量进行检测、启用调试型相机等。类似的技术曾出现于早期的游戏开发中以提供一类自动游戏演示方案。从本质上讲，游戏执行过程中可对用户输入进行捕捉，并于随后替代实际用户输入数据。然而，该方案通常包含某些特定的需求条件，从某种程度上讲，这也可视为一类缺陷。

- ☑ 捕捉时期开始阶段的初始状态。通常，可重启游戏关卡实现该过程；更为重要的是，该过程还应包含已知值并用于生成伪随机数。
- ☑ 这里，系统应具备捕捉底层控制器信息的能力，且当执行游戏回放操作时，应可将这一类数据反馈至输入处理操作中。
- ☑ 执行既定游戏逻辑，尤其是游戏外部资源发生变化时（如网络延迟、生成随机数、同步显示操作等）。
- ☑ 更新操作之间的时间间隔值应与用户输入数据一并存储。

需要说明的是，即使游戏回放机制尚不完美，但它依然不失为一类较具吸引力的解决方案。当问题难以重现或取决于特定的用户输入数据等相关条件时，回放机制则十分有效。

本章小结

相机系统与游戏体系结构以及脚本机制具有同等的重要性。尽管相机系统可独立于特定的游戏而存在，但与其他系统实现有效的整合后，其功效不言而喻。针对相机系统的实现，本章所提供的相关示例可视为一类参考方案，对应方法确实可行，其效果同样令人满意。

参 考 文 献

这里，推荐读者进一步阅读一些学术专著，相关书籍涉及相机理论、相机设计和/或对应的实现方案。其中，所列著作按内容的相似程度进行划分，某些专著则跨越多个知识领域。

AI 和漫游技术

[Borenstein90] Borenstein, J. Koren, Y. (1990).“Realtime Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments.”*The 1990 IEEE International Conference on Robotics and Automation*, Cincinnati, Ohio, May 13-18, pp. 572-577.

[Bourne05] Bourne, Owen.Sattar, Abdul.“Applying Weighting to Autonomous Camera Control.” *Proceedings of Artifi cial Intelligence and Interactive Digital Entertainment 2005*.American Association for Artifi cial Intelligence, 2005.

[Carlisle04] Carlisle, Phil.“An AI Approach to Creating an Intelligent Camera System.”In Steve, Rabin (ed.), *AI Game Programming Wisdom 2*.Charles River Media, 2004.

[Isla05] Isla, Damian.“Managing Complexity in the Halo 2 AI System.”*Presentation at Game Developers Conference 2005*, San Francisco, 2005.

[Koren91] Koren, Y. Borenstein, J. (1991).“Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation.”*Proceedings of the IEEE Conference on Robotics and Automation*, Sacramento, California, April 7-12, pp. 1398-1404.

[Millington06] Millington, Ian.*Artifi cial Intelligence for Games*.Morgan Kaufman, 2006.

[Reynolds87] Reynolds, C.W.“Flocks, Herds, and Schools:A Distributed Behavioral Model.”*Computer Graphics (Volume 21, Number 4)*:25-34.(SIGGRAPH ‘87 Conference Proceedings), 1987.

[Stone04] Stone, Jonathan.“Third Person Camera Navigation.”In Kirmse, Andrew (ed.), *Game Programming Gems 4*.Charles River Media, 2004.

[Stout04] Stout, Bryan.“Artifi cial Potential Fields for Navigation and Animation.”*Presentation at Game Developers Conference 2004*, San Jose, 2004.Written GDC 2004 proceedings.

电影技术

[Arijon91] Arijon, Daniel.Grammar of the Film Language.Silman-James Press, 1991.

[Glassner04] Glassner, Andrew.Interactive Storytelling.A K Peters, 2004.

[Hawkins02] Hawkins, Brian.“Creating an Event-Driven Cinematic Camera.”Game Developer Magazine

(Volume 9, Number 10 & 11).CMP Publications, 2002.

[Hawkins05] Hawkins, Brian.Real-Time Cinematography for Games.Charles River Media, 2005.

[Lander00] Lander, Jeff."Lights … Camera … Let's Have Some Action Already!"Game Developer Magazine(Volume 7, Number 4).CMP Publications, 2000.

[Lin04] Lin, Ting-Chieh.Shih, Zen-Chung.Tsai, Yu- Ting."Cinematic Camera Control in 3D Computer Games."WSCG 2004 SHORT Communication papers proceedings.UNION Agency — Science Press, 2004.

[Malkiewicz92] Malkiewicz, Kris.Cinematography.Simon & Schuster Inc., 1992.

[Thiery07] Thiery,Adam."Interactive Cinematography."Presentation at Game Developers Conference 2007, San Francisco, 2007. http://www.cmpevents.com/sessions/GD/S4395i1.pdf.

相机设计

[Aszódi05] Aszódi, Barnabás, Czuczor, Szabolcs."Realistic Camera Movement in a 3D Car Simulator." In Pallister, Kim (ed.), *Game Programming Gems 5*.Charles River Media, 2005.

[Ellis99] Ellis, Stephen R. Adelstein, B.D.Baumeler, S. Jense, G.J.Jacoby, R.H."Sensor Spatial Distortion, Visual Latency, and Update Rate Effects on 3D Tracking in Virtual Environments."*Proceedings of IEEE Virtual Reality Conference 1999*.

[Giors04] Giors, John."The Full Spectrum Warrior Camera System."*Presentation at Game Developers Conference 2004*, San Jose, 2004. http://www.gdconf.com/conference/2004.htm.

[Halper01] Halper, Nicolas, Helbing, Ralf, Strothotte, Thomas."A Camera Engine for Computer Games:Managing the Trade-Off Between Constraint Satisfaction and Frame Coherence."*Proceedings of EUROGRAPHICS 2001 (Volume 20, Number 3)*.Blackwell Publishers, 2001.

[Hornung03] Hornung, Alexander, Lakemeyer, G., Trogemann, G. "An Autonomous Real-Time Camera Agent for Interactive Narratives and Games."*Proceedings of Intelligent Virtual Agents*, 2003.

[Khrakar02] Kharkar, Sandeep."Camera AI for Replays."In Rabin, Steve (ed.), *AI Game Programming Wisdom*.Charles River Media, 2002.

[Khrakar04] Kharkar, Sandeep."A Modular Camera Architecture for Intelligent Control."In Rabin, Steve(ed.), *AI Game Programming Wisdom 2*.Charles River Media, 2004.

[Rabin00] Rabin, Steve."Classic Super Mario 64 Third-Person Control and Animation."In DeLoura, Mark(ed.), *Game Programming Gems*.Charles River Media, 2000.

[Treglia00] Treglia, Dante II."Camera Control Techniques."In DeLoura, Mark (ed.), *Game Programming Gems*.Charles River Media, 2000.

游戏设计

[Crawford03] Crawford, Chris.*Chris Crawford on Game Design*.New Riders Games, 2003.

[Salen04] Salen, Katie, Zimmerman, Eric.*Rules Of Play*.MIT Press, 2004.

[West05] West, Mick."Pushing Buttons."*Game Developer Magazine* (*Volume 12, Number 5*).CMP Publications, 2005.

[Wolf03] Wolf, Mark, Perron, Bernard (eds.).*The Video Game Theory Reader*.Routledge, 2003.

图形学和渲染技术

[Akenine-Möller02] Akenine-Möller, Tomas, Haines, Eric.*Real-Time Rendering*.A K Peters, 2002.

[Blinn96] Blinn, Jim.Jim Blinn's Corner:A Trip Down the Graphics Pipeline.Morgan Kaufmann Publishers, 1996.

[Demers04] Demers, Joe."Depth of Field:A Survey of Techniques."In Fernando, Rendima (ed.), *GPU Gems*.Addison-Wesley, 2004.

[Eberly04] Eberly, David H. *3D Game Engine Architecture*.Morgan Kaufmann Publishers, 2004.

[Foley90] Foley, James D., Van Dam, Andries, Feiner, Steven K., Hughes, John F. *Computer Graphics: Principles and Practice*.Addison-Wesley, 1990.

[Glassner95] Glassner, Andrew.*Principles of Digital Image Synthesis, Volume 1*.Morgan Kaufmann Publishers, 1995.

[Pharr04] Pharr, Matt, Humphreys, Greg.*Physically Based Rendering*.Morgan Kaufmann Publishers, 2004.

[Poynton02] Poynton, Charles.*Digital Video and HDTV, Algorithms and Interfaces*.Morgan Kaufmann Publishers, 2002.

数学

[Barrera05] Barrera, Tony, Hast, Anders, Bengtsson, Ewert."Minimal Acceleration Hermite Curves."In Pallister, Kim (ed.), *Game Programming Gems 5*.Charles River Media, 2005.

[Bartels87] Bartels, Richard H., Beatty, John C., Barsky, Brian A. An Introduction to Splines for use in Computer Graphics and Geometric Modeling.Morgan Kaufmann Publishers, 1987.

[Bloomenthal97] Bloomenthal, Jules (ed.).Introduction to Implicit Surfaces.Morgan Kaufmann Publishers, 1997.

[Crenshaw00] Crenshaw, Jack W. MATH Toolkit for REALTIME Programming.CMP Books, 2000.

[Eberly03] Eberly, David H. Game Physics.Morgan Kaufmann Publishers, 2003.

[Ericson05] Ericson, Christer.Real-Time Collision Detection.Morgan Kaufmann Publishers, 2005.

[Farin02] Farin, Gerald.Curves and Surfaces for CAGD.Morgan Kaufmann Publishers, 2002.

[Glassner99] Glassner, Andrew.Andrew Glassner's Notebook.Morgan Kaufmann Publishers, 1999.

[Glassner02] Glassner, Andrew.Andrew Glassner's Other Notebook.A K Peters, 2002.

[Hanson05] Hanson, Andrew J. Visualizing Quaternions.Morgan Kaufmann Publishers, 2005.

[Lowe04a] Lowe, Thomas."Critically Damped Ease-In/Ease-Out Smoothing."In Kirmse, Andrew (ed.),

Game Programming Gems 4.Charles River Media, 2004.

[Lowe04b] Lowe, Thomas."Nonuniform Splines."In Kirmse, Andrew (ed.), Game Programming Gems 4.Charles River Media, 2004.

[Melax00] Melax, Stan."The Shortest Arc Quaternion."In DeLoura, Mark (ed.), Game Programming Gems.Charles River Media, 2000.

[Ramamoorthi97] Ramamoorthi, Ravi.Barr, Alan H.(August 1997)."Fast Construction of Accurate Quaternion Splines,"Proceedings of the 24th annual conference on Computer graphics and interactive techniques, pp. 287-292.

[Rorabaugh98] Rorabaugh, C.Britton.DSP Primer.McGraw-Hill Professional, 1998.

[Schneider02] Schneider, Philip, Eberly, David H. Geometric Tools for Computer Graphics.Morgan Kaufmann Publishers, 2002.

[Sekulic04] Sekulic, Dean."Effi cient Occlusion Culling."In Fernando, Rendima (ed.), GPU Gems.Addison-Wesley, 2004.

[Shoemake85] Shoemake, Ken.Animating Rotation with Quaternion Curves.Proceedings of ACM, 1985.

[Shoemake94] Shoemake, Ken."Fiber Bundle Twist Reduction."In Heckbert, Paul S. (ed.), *Graphics Gems IV*.Academic Press, Inc., 1994.

[VandenBergen04] Van den Bergen, Gino.*Collision Detection in Interactive 3D Environments*.Morgan Kaufmann Publishers, 2004.

[VanVerth04] Van Verth, James M., Bishop, Lars M. *Essential Mathematics for Games & Interactive Applications*.Morgan Kaufmann, 2004.

[VanVerth05] Van Verth, James M. "Spline-Based Time Control for Animation."In Pallister, Kim (ed.), *Game Programming Gems 5*.Charles River Media, 2005.

其他资源

[Thomas95] Thomas, Frank, Johnston, Ollie.*The Illusion of Life:Disney Animation*.Hyperion, 1995.

在线资源

[EssentialMath] www.essentialmath.com

[Gamasutra] www.gamasutra.com

[GDC] www.gdconf.com/archives

[Lionheart] www.robinlionheart.com/gamedev/genres.xhtml

[RealTimeRendering] www.realtimerendering.com

[Wikipedia] www.wikipedia.org

附录A 术 语 表

- ☑ 30° 规则（30 degree rule）：电影技术规则，用于限制跳转镜头的最小角位移。
- ☑ 失真变形（Anamorphic）：一类渲染方案，可用于生成扭曲画面并于随后通过镜头或投影方案加以修正。
- ☑ 宽高比（Aspect ratio）：视口或显示设备宽度和高度之间的比值。例如，全帧 SDTV 的宽高比定义为 4:3 或 1.33；而宽屏的宽高比则定义为 16:9 或 1.78。
- ☑ 吸引对象（Attractor）：此类对象（特定对象类型）向其他游戏对象施加作用力，进而辅助对象在复杂场景环境中的漫游操作，或调整当前游戏体验。其中，作用力大小通常与对象间的距离、相对方向抑或移动方向相关。
- ☑ 轴角（Axis angle）：方向的数学表现形式，通常由方向向量或围绕该方向轴的角旋转操作构成。
- ☑ 轴测投影（Axonometric projection）：正交投影的子类，其中，投影方向与视见对象轴之间并未保持对齐。对象的 Up 轴通常呈现为垂直方向，而其他两轴则呈现为不同程度的倾斜状态。此处，轴测投影包括 3 种类型，即等轴测投影、二等角投影以及三等角投影。
- ☑ 偏航角（Bank）：参见术语“倾侧旋转”。
- ☑ Bartels-Kochanek 样条（Bartels-Kochanek spline）：参数曲线形式之一，其特征类似于 Catmull-Rom 样条，其中，曲线途经全部控制点。各控制点处的参数用于控制邻接点之间的曲线形状，包括偏移量、强度以及连续性。
- ☑ Bezier 曲线（Bezier curve）：Hermite 曲线形式之一。
- ☑ 吊杆拍摄（Boom）：电影摄像机的垂直移动，通常作为部分升降机镜头加以展现。
- ☑ B 样条（B-spline）：源于三次多项式的一类曲线，且呈现为由控制点构成的凸体形状。同时，B 样条曲线无法保证该曲线途经全部控制点。
- ☑ 斜二测投影（Cabinet projection）：斜投影形式之一，其中，后退轴的比例系数是其他两轴的二分之一。
- ☑ 相机行为（Camera behavior）：相机属性集合，进而定义了游戏机场景相机的位置和方向。
- ☑ 相机抖动（Camera bob）：相机抖动多出现于第一人称相机中，通过调整与角色之间的相对位置，可极大地增加角色在游戏场景中移动时的临场感。相机抖动通常由固定的移动形状构成，其量值则随玩家角色的移动速度而变化。
- ☑ 相机提示对象（Camera hint）：该对象根据当前游戏投影或相关美学条件动态控制或调整当前相机的行为。
- ☑ 相机管理器（Camera manager，CM）：基于游戏系统的实现方案，相机管理器可视为游戏实体，并负责激活/禁用相机以及运行期内的各类应用。
- ☑ 相机震动（Camera shake）：游戏相机渲染位置处的小范围（随机）移动。此类移动方式常用于模拟真实世界中的震动现象，如爆炸效果、地震等。

- ☑ 相机系统（Camera system）：该系统负责处理全部基于游戏相机的相关事宜，并常与其他游戏系统结合使用。某些相机系统还包含视口定义功能项以及游戏场景的渲染操作。
- ☑ 场景类型（Camera type）：参见术语“相机行为”。
- ☑ Catmull-Rom 样条（Catmull-Rom spline）：三次多项式样条曲线形式之一，该样条确保曲线途经全部控制点。
- ☑ 斜等轴测投影（Cavalier projection）：斜轴测投影形式之一，且各轴上的比例系数皆相等。
- ☑ 角色相关（Character-relative）：由游戏对象（如玩家角色）方向所定义的参考帧，且与当前相机的位置和方向无关。
- ☑ 宽银幕电影（Cinema scope）：非交互式电影表现形式之一，其视口宽高比为 1.85:1。
- ☑ 电影摄像机（Cinematic camera）：通过游戏引擎表现非交互式电影视频序列的相机。
- ☑ 写实电影（Cinéma verité）：电影剧作中追求逼真地、如实地描绘客观现实的方法。写实电影源于纪录片并追求一种自然的拍摄效果，例如，启用非专业演员、通过手持设备进行拍摄以及其他强调影片真实性的技术手段。
- ☑ 圆周移动射击（Circle strafing）：反向旋转时，相对于玩家角色的侧向移动行为（如射击）。此类组合结果通常以圆弧方式移动，其半径多与射击范围相关。
- ☑ 碰撞器（Colliders）：该方案可辅助相机实现受限空间内的漫游操作。
- ☑ 颜色淡化（Color fade）：场景切换方式之一，其中，可视图像在一段时间内褪至某一固定颜色（淡出）。与其相反的例子则是淡入效果。
- ☑ 约束条件（Constraint）：约束条件将对相机的相关属性加以限定，进而满足特定的游戏体验以及美学需求。例如，可限制相机的纵向高度值，进而避免相机与世界 Up 轴平行。
- ☑ 控制点（Control points）：控制点可视为设置于游戏场景内的相关标记，并可用于定义和控制样条曲线的曲率。
- ☑ 控制参考帧（Control reference frame）：控制参考帧定义了控制器输入以及对象移动之间的关系。在第一人称相机中，输入内容直接映射至角色的转向或移动行为之上（有时也称作角色相关）；在第三人称相机中，该过程则取决于玩家角色与相机之间的相对位置（也称作相机相关）。
- ☑ 升降机镜头（Crane shot）：针对于真实摄像机，该类镜头需要使用到物理升降臂以在镜头拍摄过程中升降、移动相机。
- ☑ 剔除操作（Culling）：该操作根据视锥体的尺寸确定场景渲染元素。
- ☑ 旁跳镜头（Cut-away shot）：旁跳镜头并非展示当前行为动作，而是追回至原场景画面。旁跳的对象可以是迥然不同的事物、局部特写抑或其他场景。
- ☑ 过场动画（Cut scene）：参见术语“非交互式电影”。
- ☑ 阻尼机制（Damping）：通过减少或消除位移量以生成平滑的加速和减速效果。
- ☑ “死亡”相机（Death camera）：玩家角色死亡时的表现方式，通常采用相对于角色的外部视口加以显示（并非必需）。“死亡”相机在角色死亡之前复制游戏的近期内容，并进一步强调与玩家死亡相关的敌方角色或事件，而非简单地将注意力集中于角色之上。鉴于玩家在死亡时段通常无法执行任何动作，因而可通过各类视频技术适当地展现这一过程。
- ☑ 时间间隔（Delta time）：自前次更新操作后的时间流逝量。
- ☑ 深度缓冲区（Depth buffer）：系统或图形内存区域，用于确定显示设备各像素处的最近对象深度信息，深度缓冲区多出现于渲染操作中，进而判断新像素是否可执行渲染操作。

- ☑ 景深（Depth of field，DOF）：相机焦点附近处的距离，在此距离范围内，对象将保持清晰的聚焦状态。
- ☑ 期望方向（Desired orientation）：游戏相机的视见方向和旋转操作且通常在世界坐标空间中加以定义。此处，可通过隐式的游戏相机旋转操作以使其匹配于游戏场景的 Up 轴。
- ☑ 期望位置（Desired position）：相机行为产生的理想位置。
- ☑ 斜测投影（Dimetric projection）：轴测投影形式之一，其中，两轴具有类似的缩放比例，而第三轴则可独自加以定义。
- ☑ 显示场（Display field）：针对于隔行显示，显示场可视为一段时间值，其中，交替变化的像素行将被显示。针对 NTSC 以及 MPAL 显示制式，该值为 1/60 秒；而 PAL 以及 SECAM 制式，则该值为 1/50 秒。
- ☑ 显示帧（Display frame）：类似于显示场（针对逐次逼近显示），显示帧也可视为一段时间值且与游戏场景的生成过程相关。在隔行显示中，该时间值等同于两个连续的显示场时间值。
- ☑ 显示撕裂（Display tearing）：更新帧缓冲区所导致的图像负面效果，其中，图像的生成结果源于同一内存区域。在两次连续的更新操作过程中，将生成不连续的分裂画面。
- ☑ 跟踪拍摄（Dolly）：该电影术语描述了相机朝向或远离目标对象的移动行为。通过该操作，相机可平滑地穿梭于场景间并对相关对象实施有效的跟踪。
- ☑ 多普勒效应（Doppler effect）：当相对于声源移动时，该术语描述了频率的变化状态，如警笛。
- ☑ 双缓冲区（Double buffering）：该方案可有效地避免显示撕裂现象，并通过以下方式提高图形性能，即在显示当前渲染图像时，可连续渲染第二幅图像。
- ☑ 俯仰角度（Dutch tilt）：镜头摇动方式之一，有时也称作 Dutch 角。
- ☑ Ease 函数（Ease function）：插值计算过程中的特定函数，其中，开始值和目标值之间的加速和减速状态可通过该函数在一段时间内的评估值加以确定。
- ☑ 欧拉角（Euler angles）：针对于已知参考帧（如世界坐标方案），该术语定义了方向的数学表达方式。欧拉角体现了围绕 3 个轴向的旋转操作，且经累积后最终生成结果方向。需要注意的是，欧拉角计算不满足交换律，因此，对应的计算顺序则显得十分重要。
- ☑ 安全检测条件（Fail-safe）：该术语源于程序设计，并用于游戏相机的重定位（以及重定向)操作，以防止游戏机场景出现不适宜的画面内容。根据游戏特征，条件检测过程可通过多种方式加以触发。常见的检测内容包括目标对象的遮挡现象、相机是否位于游戏场景外部空间等。
- ☑ 远剪裁面（Far clipping plane）：该术语源于渲染操作且定义了与相机之间的距离值（位于局部空间内），位于该距离值范围之外的对象将不予渲染。远剪裁面平行于近剪裁面并构成了视锥体的边界。远剪裁面有时也称作后平面，或简称为远平面。
- ☑ 视域（Field of view，FOV）：用于连接视口宽高比的角度值，进而确定视锥体的可视范围，对应角度包括纵向垂直角度以及水平角度。
- ☑ 有限状态自动机（Finite state machine，FSM）：有限状态自动机可视为一类规划方案，并对游戏对象的状态加以管理。通常情况下，FSM 由多个状态和触发器构成，其排列方式多根据所需实体行为加以设置。当满足相应的触发器条件时，状态间将实施转换操作。
- ☑ 第一人称（First person，FP）：FP 可视为一种表现风格，其中，游戏场景的视见内容将根据角色的眼部位置加以定位。
- ☑ 固定着色（Flat shading）：一种渲染方案，其中，各多边形对象表面均采用固定单一颜色加以

绘制。

- ☑ 流场（Flow field）：流场多见于游戏 AI 领域，以辅助对象在复杂的场景环境中实现径直移动。
- ☑ 强行透视（Forced perspective）：强行透视是一类渲染技巧，当采用该技术时，与对象的真实场景位置相比，其距相机之间的距离将呈现为不同的状态。鉴于远剪裁面的限制，当针对远处对象进行渲染时，可适当地考虑使用此类技术。
- ☑ 帧速率（Frame rate）：一种相对简单的性能测算技术，有时被视屏游戏作为一种量化方法加以引用。另外，帧速率可视为单位时间内（以秒计）更新操作的执行次数，有时定义为单位时间内的帧数量（FPS）。
- ☑ 取景操作（Framing）：一种定义相机位置和方向的方法，以满足期望中的视觉条件，如相机视口内的游戏元素的定位和尺寸。
- ☑ 全自由视角（Free look）：该术语是指玩家针对游戏相机方向的操控能力。全自由视角多出现于第一人称相机中，第三人称相机对此也有所涉及。另外，相对于玩家角色或其他固定帧，后者常与相机移动结合使用。
- ☑ 视锥体（Frustum）：视锥体呈金字塔状且包含矩形底面，其中，顶端部分被平行于底面的平面所截取。此类视锥体常用于形成剪裁面，进而可快速判断可渲染的场景元素。针对视锥体中的两个平行平面，距相机位置较近的平面称作近剪裁面；而较远的平面则称作远剪裁面。另外，位于视锥体外部的对象将不对其执行渲染操作。
- ☑ 全屏显示（Full frame）：该术语是指采用全屏幕渲染游戏场景或视频序列。另外，全屏显示多与宽高比为 4:3 的显示设备相关。
- ☑ 游戏相机（Game camera，GC）：包含各种属性的游戏实体，用于定义位置、方向以及游戏场景其他需求条件。
- ☑ 万向节死锁（Gimbal lock）：当出现万向节死锁现象时，玩家将无法对相机与玩家对象间的位置和方向实施有效的操控。另外，数学运算所导致的异常将使得相机方向围绕其前视向量呈现为快速变化。
- ☑ 硬约束条件（Hard constraints）：相机需强制执行的一类限制条件（如基于移动或方向的限定条件），进而可防止出现渲染几何体相交这一类现象。
- ☑ 仰视视角显示系统（Heads-up display，HUD）：该术语源于航空学，在游戏中，HUD 则指屏幕中的一类覆盖元素，以向玩家提供额外的游戏信息。这里，常见的 HUD 元素包括瞄准镜（或鼠标指针）、玩家状态信息、地图、分值状态和游戏场景方位等。
- ☑ Hermite 曲线（Hermite curve）：三次多项式样条曲线形式之一。
- ☑ 高清电视（High definition television，HDTV）：HDTV 标准其水平分辨率为 1920 像素；垂直分辨率为 1080 像素（NTSC 标准），并采用隔行或步进生成显示技术。该术语有时常被误用为垂直方向 720 像素。
- ☑ Hither 平面（Hither plane）：近剪裁面的别称。
- ☑ 混合相机方案（Hybrid camera）：基于第一人称相机和第三人称相机的游戏相机。
- ☑ 滞后现象（Hysteresis）：使用与相机属性相关的历史信息，进而减少当前属性中的震动因素。
- ☑ 闲置操作（Idle wandering）：在第一人称游戏中，待一段时间内的休止状态后，或缺少玩家输入操作时，常会引入相机的半随机重定向操作。此类方向变化类似于闲置动画，其中，玩家角色将会采用相关动画效果以显示当前角色处于等待玩家输入状态。

- ☑ 影响力地图（Influence maps）：影响力地图可视为一类漫游辅助手段，进而使得设计人员可定义作用于相机或 AI 移动之上的影响力，以此躲避游戏机场景中的相关元素。
- ☑ 插入镜头（Insert shot）：插入镜头源于电影术语，即源于不同角度或焦距的一类镜头。插入镜头不同于旁跳镜头，其中，当前镜头依然关注于同一事物或场景。
- ☑ 交互式操作（Interactive）：交互式操作发生于游戏体验过程中，其中，玩家尝试控制其角色或通过其他方式与游戏元素进行交互。
- ☑ 隔行扫描（Interlacing）：隔行扫描可视为一类显式技术，旨在减少显式设备发射出的水平扫描线数量，且奇偶扫描线交替出现。此处，各奇偶扫描线集称作扫描场，针对于 NTSC 制式，扫描线每 1/60 秒奇偶交替出现。因此，同一时间内可生成 30 个全屏幕帧。
- ☑ 插值计算（Interpolation）：插值计算（亦称作混合计算）通常涉及动画帧的生成、位置间的对象移动行为、颜色值调整以及各种类似的操作，其实现方法亦不一而同。
- ☑ 等轴测投影（Isometric projection）：由二维元素构成的伪三维表现形式。等轴测表现方式可视为一类平行投影，其中，测轴与水平方向呈 30° 角。另外，各轴均采用相同的比例系数。
- ☑ 跳转操作（Jump cut）：镜头跳转可视为一类电影技术规则，此时，呈现于观察者眼中的场景将产生瞬时变化。通常情况下，活动相机的位置将发生较大的改变，并展现完全不同的场景且不会向观察者发出任何提示（如画面交接或其他切换效果）。除此之外，相机在保持静止的同时创建可发生迅速的变化（例如，相机方向逆置、保持同一位置和方向，而场景呈现为昼夜变化等）。
- ☑ 关键帧（Key frame）：该术语源于动画操作，即基于插值中间值的数据集合（如对象的位置和/或方向数据）。另外，若原始数据采样率较大并可生成较为平滑的变化值，则可直接使用关键帧数据。
- ☑ 关键点（Keys）：针对于样条曲线，该术语即为控制点。
- ☑ 延迟效应（Lag）：如果相机属性（如方向和位置）的更新操作稍慢于其变化率，则可生成具有一定美学效果的画面结果。
- ☑ 镜头光晕（Lens flare）：当相机镜头的摄入光线过于强烈时所导致的一类画面效果。该效果可运用于视频游戏中，以增强画面的真实感。
- ☑ 视线（Line of sight，LOS）：场景中的投射直线，进而可判断对象间是否可见。视线常用于确定玩家角色与当前（或未来）相机位置之间是否可见。
- ☑ 锁定（Lock-on）：在多数第一人称游戏以及某些第三人称游戏中，玩家角色可相对于目标对象或其位置进行移动，并保持其方向面向目标对象。因此，当前角色围绕目标位置呈现为圆弧运动，则该角色处于“锁定”状态。针对玩家而言，此类移动方式也称作基于轨道方式的旋转运动。
- ☑ 视见位置（Look-at position）：该术语定义了游戏相机在游戏场景中的位置，进而确定相机的方向。视见位置多针对游戏对象加以计算，并根据游戏体验需求或对象状态变化做适当调整。
- ☑ Macro 贴图（Macro tiles）：在定义二维游戏中的背景图案且存储重复模式的贴图时，该方案可有效地降低所占用的内存空间。
- ☑ 管理器（Manager）：管理器可视为一类程序结构并用于控制实时应用程序中的某些特定模块，如相机系统管理。
- ☑ 运动模糊（Motion blur）：后期图像中的模糊状态，该效果通常由胶片感光过度或快速移动的

对象所引起。

- ☑ 近剪裁面（Near clipping plane）：该术语有时也称作近平面，近剪裁面定义了视锥体最近表面距相机位置之间的距离。小于该距离范围的相关对象将不予渲染；同时，与近剪裁面相交的对象将对图形效果带来负面影响，如残缺不全的多边形表面渲染效果等。
- ☑ 对象相关（Object-relative）：坐标系统原点根据游戏对象的位置和方向加以测算。
- ☑ 斜投影（Oblique projection）：轴测投影形式之一，其中，两个轴平行于投影平面，第三个轴则与水平方向呈45° 或60° 倾斜，斜二轴测投影以及斜等轴测投影便是此类投影的常见形式。
- ☑ 遮挡效果（Occlusion）：在相机系统中，鉴于场景环境特征以及位于相机及其目标之间的游戏对象，遮挡效果体现了玩家角色或其他目标对象的视见隐匿状态。短期内的遮挡效果尚可令人接受，否则，应对此实施相应的条件检测。
- ☑ 轨迹锁定（Orbit-lock）：参见术语“锁定”。
- ☑ 方向（Orientation）：该术语定义了游戏实体与参考帧之间的方向问题。通常情况下，可采用世界坐标轴定义绝对方向。
- ☑ 正交投影（Orthographic projection）：采用平行于世界轴或模型轴的方向生成游戏场景的视见内容，正交投影多用于创建屏幕地图。当采用正交投影时，游戏场景内的平行直线将保持原有状态。
- ☑ 调色板循环机制（Palette cycling）：一类简化的动画技术，根据既定颜色值的调色板而非某一特定颜色，可通过不同颜色值渲染部分场景。颜色调色板的各项数据均按一定顺序加以设定，其中，临近像素与调色板的邻接项实现彼此匹配。通过复制调色板各个数据项，绘制像素的移动方向将与调色板的复制顺序相吻合。需要说明的是，该技术在很大程度上可被纹理动画所替代，例如可在一段时间内简单地改变纹理坐标值，进而实现同一效果。
- ☑ 镜头上移（Panning）：相机移动垂直于当前方向，同时保持相机方向不变。
- ☑ 视差（Parallax）：根据距相机位置间的距离值，渲染场景元素的相对移动，例如2D卷轴游戏中的图层。
- ☑ 路径相机（Path camera）：采用预定义路径（如样条曲线）的相机，进而确定游戏机场景中的相机位置。
- ☑ 路径搜索（Path-finding）：复杂场景中辅助漫游技术的统称，相关算法可判断连接位置或区域间遍历行为的有效性（以及开销）。
- ☑ 透视投影（Perspective projection）：虚拟场景世界的构建方法，其中，平行线汇聚于一点，且对象根据相机视点呈现为“近大远小”状态。游戏对象作为几何体图元集合（通常为三角形或四边形）加以构建，且于随后投影至二维表面之上。该投影操作与人类视觉系统对真实世界的感观方式类似。
- ☑ PID控制器（PID controller）：比例积分反馈控制器多用于过程功能处理行业以及相关应用程序，例如自动调温器和防抱死制动系统。相应地，相机系统所采用的数字化版本则向相机属性（如相机移动行为）提供了对应的阻尼功能。
- ☑ 玩家角色（Player character）：玩家角色代表了虚拟环境中的游戏对象，常用于定义相机的期望位置以及期望方向。有时，这一类角色也被认为是玩家在游戏中的化身。
- ☑ 视点（Point of view）：视点常用于第一人称相机中，即主角色或其他游戏角色的眼部观察位置。
- ☑ 势场（Potential fields）：势场可视为一类程序设计方案，用以模拟游戏环境中对象偏离障碍物

的移动方向。

- ☑ 预测相机（Predictive camera）：这一类虚拟相机可用于估测与目标数据相关的未来信息（如位置数据）。
- ☑ 投影（Projection）：该操作可实现不同形式的坐标转换。较为常见的游戏相机系统将世界空间坐标分别投影至相机空间以及屏幕空间（通过相应的透视转换）。
- ☑ 投影转换（Projection transform）：投影转换是一类数学转换过程，并将某系形式的空间坐标信息（如世界空间坐标）转换至另一空间中（如用于渲染操作的相机空间）。
- ☑ 比例控制器（Proportional controller）：针对于易入操作（ease-in），比例控制器可视为一类线性反馈控制系统，进而可消除基于目标值的过冲现象。当采用比例控制器时，控制器输出与当前值和目标值之间的变化呈某种比例状态。
- ☑ 四元数（Quaternion）：四元数可视为一类数学运算过程，用于表现对象的旋转操作。针对插值以及其他角度操作，四元数可有效地避免万向节死锁等问题（该问题多出现于采用欧拉角这一类表达方式时）。
- ☑ 光线投射机制（Ray casting）：针对游戏场景数据结构的一类查询方法，进而判断两点是否存在无遮挡线性路径。
- ☑ 特写镜头（Reaction shot）：特写镜头源于电影拍摄术语，用于描述角色特写部分的切换镜头，进而表现角色对外部事件所产生的反应。
- ☑ 反馈相机（Reactive camera）：根据目标对象的当前或历史状态定义其行为的一种游戏相机。
- ☑ 真实相机（Real-world camera，RWC）：物理相机设备，用于捕捉一系列的静态画面，并依次进行察看进而生成动画效果。
- ☑ 刷新率（Refresh rate）：即更新率。
- ☑ 重定向（Reorientation）：相机在一段时间内的方向变化。
- ☑ 回放相机（Replay camera）：一类虚拟相机，用于复制之前游戏场景中的位置和方向序列。相关位置数据可采用多种方式加以生成，进而获取对应的游戏场景镜头或视频序列。
- ☑ 排斥体（Repulsor）：排斥体可向其他游戏对象施加作用力，以辅助该对象实现复杂场景中的漫游操作。其中，作用力的大小可根据排斥体类型做适当调整，但通常与对象至排斥体之间的距离呈某种比例关系（有时也取决于对象和排斥体之间的相对移动方向）。
- ☑ 倾侧旋转（Roll）：相机围绕前视方向的旋转操作，多数时候，该操作限定于非交互式视频序列。倾侧旋转操作常见于车辆类游戏中，并在一定程度上增强玩家的临场感。需要注意的是，过度使用该效果将导致玩家产生某种不适感。
- ☑ 安全区（Safe frame）：该术语是指显示设备中的绝大部分有效区域，进而向观察者显示相关信息，且不考虑当前显示设备的物理特性（如屏幕曲率）。
- ☑ 安全框（Safe zone）：位于安全区之外的显示设备的边框区域。
- ☑ 脚本机制（Scripting）：该术语定义了一类相关机制并用于控制游戏事件，并使其出现于特定的时间点，抑或对其他游戏事件加以反馈。
- ☑ 着色语言（Shader language）：基于图形管线的底层程序设计语言。着色器可调整纹理的绘制方式，并可有效地刻画真实世界的图形特征，如粗糙度等。
- ☑ 横向卷轴（Side-scrolling）：源于视点的一类游戏场景透视表现方式，且垂直于玩家角色的移动方向。

- ☑ 天空盒（Sky box）：可用于构建远距离游戏场景的静态几何体，例如天空景观或其他远方场景元素（如远山）。通常情况下，天空盒多采用简化的几何体并与纹理贴图结合使用，并在其他场景元素之前加以渲染。
- ☑ 快照相机（Snapshot）：此类相机有其自身在场景中的位置和方向数据，且多用于下列情形，即全场景渲染操作具有较大的实施计算开销，系列游戏 Resident Evil（亦即 Biohazard，生化危机）即采用了此类相机模式。在这一类游戏中，背景图案仅渲染一次并于随后加以存储（包括深度排序信息）；在游戏执行过程中，仅游戏对象在当前场景内加以渲染，并与前次计算后的深度信息结合使用。
- ☑ 软约束条件（Soft constraints）：附加在相机之上的约束条件（如移动和方向限制条件）且不需强制执行。
- ☑ 软锁（Soft-lock）：鉴于未知环境状态，玩家无法在游戏环境中执行进一步操作，如锁定后的门对象无法正常开启。
- ☑ 转轴（Spindle）：术语“轴”的别称，进而向相机移动或其他计算属性提供相应的参考帧。
- ☑ 样条（Spline）：三次多项式曲线的别称。
- ☑ 分屏（Split screen）：将显示设备划分为多个显示区域，分屏机制多用于单显示设备上的多玩家游戏，其中，玩家包含各自对应的显示区域。除此之外，该模式也适用于单玩家游戏的多场景视口操作，抑或简单地向玩家显示相关信息（如文本信息、后视镜等）。
- ☑ 弹性机制（Spring）：该机制支持游戏对象平滑的加速以及减速操作。其中，游戏系统弹性机制将施加严格意义上的阻尼效果，进而防止过冲现象的出现，并消除静止位置附近处的振荡行为。
- ☑ 精灵对象（Sprite）：精灵对象可视为一类二维图像且多出现于 2D 风格的游戏中，尽管某些 3D 游戏也采用精灵对象并将其视为低开销方案加以使用。针对于前者，多数游戏元素均采用此类对象加以显示；另外，大多数早期游戏仅支持基于精灵对象的显示方案。在三维表现方式中，精灵对象常用于表现以下对象，即其方向相对于当前相机变化较少。考虑到缓存以及复用特性，精灵对象可有效地降低系统的性能开销，尽管其视觉效果稍有下降。虽然多数精灵对象被视为一类二维对象，但依然可对其深度信息加以存储，进而可在其渲染过程中使用对应的深度排序数据。
- ☑ 标清电视（Standard definition television，SDTV）：基于 SDTV 的显示设备其宽高比为 4:3，垂直分辨率为 625 条扫描线（PAL 制式）或 525 条扫描线（NTSC 制式）。
- ☑ 贴花图案（Stamp）：参见术语“贴图”。
- ☑ 目标对象（Target object）：目标对象定义为一类游戏实体并用于确定游戏相机的属性。目标对象最为常见的应用是定义相机的期望方向，某些情况下，目标对象相对于参考帧的期望位置也可用于动态调整相机的各类属性。
- ☑ 目标位置（Target position）：术语“期望位置”的别称。
- ☑ 第三人称（Third person）：第三人称体现了游戏的表现风格，其中，游戏场景视点定位于当前玩家角色的外部。
- ☑ 贴图（Tile）：二维游戏图形元素（类似于精灵对象），且通过阵列或均匀网格的方式加以渲染。其中，各贴图均由像素数组构成且尺寸固定（如 8×8、8×16、16×16 等）。另外，贴图网格也称作纹理贴图，并可通过索引对贴图集加以访问。因此，基于纹理图的贴图复用模式无须

额外的贴图定义信息。贴图的全部数量定义了纹理图内的索引值尺寸。需要注意的是，纹理贴图内的相关元素无须彼此邻接；另外，各贴图尺寸亦无须保持均等（早期 8 位或 16 位游戏则对此有所限制）。

- ☑ 倾斜（Tilt）：术语“倾侧旋转”的别称。
- ☑ 转换矩阵（Transformation matrix）：游戏对象的方向数据（以及位置数据）的数学表现方式，通常由 3×3、3×4 或 4×4 定点或浮点数组构成。
- ☑ 渐变切换（Transition）：不同表现方式之间的转换操作，通常指第一人称至第三人称之间的转换，反之亦然。该操作可视为插值计算中的一个特例，尽管某些时候会产生不连续现象。
- ☑ 三度投影（Trimetric projection）：轴测投影形式之一，其中，3 个轴均包含彼此独立的比例系数。
- ☑ 三缓冲机制（Triple buffering）：三缓冲机制可视为一类有效的程序设计技术，通过图像渲染硬件与中央处理器之间的并行操作，该方案可有效地增加系统的数据吞吐量。其中，连续的渲染操作将使用不同的内存区域（以循环方式），进而可有效地降低系统的等待时间（前次渲染图像缓冲区显示完毕）。
- ☑ 转向（Twist）：目标对象在相机上方或下方移动时所导致的游戏相机的快速重定向操作，该情形通常不可取。
- ☑ 更新循环（Update loop）：游戏引擎的主要组成部分，负责更新游戏中的各项元素。
- ☑ 更新率（Update rate）：更新全部游戏逻辑的时间值，通常以更新次数/秒计算。
- ☑ 垂直转向（Vertical twist）：当目标对象接近相机时（在水平面中）相机的快速重定向操作，这将导致玩家的混乱感，进而难以对当前玩家角色实施有效的控制。
- ☑ 视锥体（View frustum）：由剪裁面定义的空间体，进而定义了游戏场景的可渲染部分。
- ☑ 视口（Viewport）：显示设备的部分区域，用于渲染游戏场景的视见画面。
- ☑ 视见转换（View transform）：针对于渲染操作，视见转换定义了相关的转换矩阵，并将坐标由相机空间转换至屏幕空间。
- ☑ 视见体（View volume）：术语“视锥体”的别称。
- ☑ 空间体投射（Volume casting）：基于三角表达方式的三维形状投影方案（如游戏场景内对象的移动行为）。
- ☑ 宽屏（Wide screen）：宽高比为 16:9 的显示设备。
- ☑ 世界相关（World-relative）：基于标准参考帧（而非虚拟场景中对象的方向）的坐标系统。
- ☑ 仰角（Yaw）：相机系统的水平定向，并通过围绕世界 Up 轴的旋转操作加以定义。
- ☑ Yon 平面（Yon plane）：近剪裁面的别称。
- ☑ Z 缓冲区（Z-buffer）：Z 缓冲区（或称作深度缓冲区）存储了帧缓冲区中各渲染像素的深度信息。这里，可利用 Z 缓冲区中的既定值与绘制像素的计算值进行比较，进而渲染过程将剔除被近相机对象所遮挡的相关像素。

附录B 辅助材料

视觉感知

人类视觉感知系统涉及了丰富的内容，加深对基础知识的理解将有助于实现良好的相机系统，并针对相关游戏场合选取正确的应用方案。这里，关于人眼、光线与电脉冲之间的转换方式，抛开其本质特征不谈，物理以及心理影响也会改变人类对当前环境的理解方式（严格意义上讲则是感知方式）。

某些情况下，鉴于某些自然因素（如伪装效果）或由于缺乏相应的参考帧（如强迫透视），这一类技术无法生成相对准确的信息。上述技术可用于虚拟场景的构建过程中，进而可极大地提升玩家的真实感，抑或调整观察者对虚拟环境的感知程度。

移动敏感度

首先，应强调人类视觉系统针对移动行为的敏感度，以及大脑对此类感观数据的传递方式。这里，人类对移动行为的感知可通过多种方式加以确定，如相关视觉线索。另外，内耳中的体液流动亦针对移动以及方向提供了重要的信息。除此之外，声音线索也包含了丰富的移动信息（环绕声或立体声，如多普勒效应）。尽管视频游戏向玩家提供了丰富的视觉以及声音信息，然而该过程并不包含任何内耳信息。由于玩家注视区域的外部环境缺乏应有的移动行为（假设显示设备的尺寸以及观察距离均处于正常范围内），当玩家大脑接收某些不一致的信息时，常会导致玩家产生某种不适感。另外，在某些第一人称游戏中，为了改善有限的场景视野，相关视觉元素还将会进一步加深这种不适感，如震动效果。此时，相机的渲染位置或玩家的武器装备应适时移动，以配合当前快速的移动行为，如玩家角色应处于奔跑状态。通常，实际运动行为常体现为少量的纵向移动（近似定时移动，以匹配玩家角色的脚步与地面之间的接触动作）抑或沿屏幕空间内的既定样条或路径进行移动（如“8”字形路线）。

实际上，这并非是自然状态下的移动行为，且仅用于强调玩家的相关动作而已。通常，可通过相机移动以对游戏场景中的其他物理现象进行适当的反馈，如玩家跳跃或跌倒时的着陆状态、武器装备的反作用力、爆炸效果以及玩家角色的其他受力状态。

刷新敏感度

另一类问题则是渲染场景的更新率。考虑到各类变化（尤其是刷新率降低时）将会分散玩家的注

意力，因而理想状态下，更新操作应锁定于某一固定频率上；针对于电视系统加以设计的应用程序，同步状态可根据对应的电视制式加以制定（例如，PAL 制式或 SECAM 制式为 50Hz，NTSC 制式或 PAL60 制式为 60Hz）。而基于个人计算机以及显示器的应用程序则无须实现同步操作。实际上，即使采用电视系统加以显示时，此类同步操作往往亦忽略不计，进而获取较高的刷新率。显示刷新率中的延迟现象（与游戏逻辑更新操作、对象移动等内容相比）将对画面显示产生较为明显的负面影响，重则还可导致玩家的极度不适应感。针对于头戴式显示设备，NASA 已对此类延迟现象进行了较为深入的研究（如[Ellis99]）。针对于第一人称视点型游戏，该方式所产生移动问题则屡见不鲜，其原因在于，结合前述问题，玩家的行为动作将导致渲染场景产生大幅度的变化。第三人称相机也会整体改变渲染场景的当前状态，但相机的移动幅度通常较小（除非相关对象途径相机位置，并导致相机的垂直转向，具体内容可参考第 7 章）。

即使渲染过程降至某一较低频率，玩家控制的反馈操作依然需要保持在较高的频率之上。如果玩家操控与角色移动之间不一致，将使玩家失去原有的方向感。

色彩敏感度

如[Glassner95]所述，人眼对色彩的敏感程度将依据色谱中的频率产生不同的变化。例如，与红色相比，蓝色则更易产生某种深远感。

对象间的色调差异可用于强调渲染画面中的相对深度关系，艺术家对此则有着更为深刻的理解。另外，类似的行为也存在于真实世界中的大气环境中。

深度感

由于视频游戏通常会将三维世界场景投影至二维平面之上，因而玩家的深度感常取决于投影之间的关系。针对于此，游戏场景投影画面内的、对象间的相对位置对于环境以及游戏对象间的漫游和交互操作均起着至关重要的作用。同时，针对物理控制器以及玩家角色的虚拟移动控制之间的关系，这一类位置关系还会对此产生深刻的影响。某些玩家则对此类关系难于理解，尤其是当游戏场景视见画面源于玩家角色之外的某一位置时。

这里，存在多种方法可提升深度感，进而强调渲染视见画面内的、对象间的相对位置关系。针对于此，可借鉴于某些电影拍摄手法。

另外，光线效果也可视为提升深度感的重要手段，具体内容可参考[Pharr04]。光线与阴影效果密不可分，亦即，对象的阴影投射常根据游戏机场景内的光源方向和位置加以设定。多数时候，考虑到真实光线方案所产生的性能开销，阴影的渲染结果仅可视为一类近似解决方案。无论如何，简单的阴影效果亦可向玩家提供较为明显的深度感，早期的处理方案则针对各类对象均采用一类柔和的阴影图案，即在游戏对象的下方表面上渲染模糊的圆状图案。

若可对阴影的表现方式进行适当的调整以使其与当前环境特征实现良好的匹配，则可大大增强对象与其场景间的“附属感”。具体而言，阴影可向玩家提供较为明显的位置深度信息，针对于悬浮对象尤是如此。当玩家角色悬空时，多数玩家均难以针对其他对象实施有效的定位，而阴影则可对此提供有效的空间信息。除此之外，阴影还可向场景添加一类视差效果，其中，根据对象与相机之间的距离，对象的相对移动方式将发生较为明显的变化；同时，视差效果将显著地提升场景画面的深度感。另外，相机附近处的前景对象（尤其是相机移动路径采用预定义方案时）应进一步强调其视差效果，但应避免出现遮挡问题。